Sun Chili!Soft ASP 3.6.2

Product Documentation

Legal Notice

Copyright 2002 Chili!Soft, Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, California 94303, U.S.A. All rights reserved. Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more additional patents or pending patent applications in the U.S. or other countries. This product documentation and the technology it describes are distributed under licenses restricting their use, copying, distribution, and decompilation. No part of this product documentation may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any. Third-party software, including font technology, is copyrighted and licensed from Sun suppliers. Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd. Sun, Sun Microsystems, the Sun Logo, Java, Solaris, and Chili!Soft ASP are trademarks or registered trademarks of Chili!Soft, Inc. or Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks of SPARC International, Inc. in the U.S. and other countries. Federal Acquisitions: Commercial Software – Government Users Subject to Standard License Terms and Conditions.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Contents

Introduction: About This Documentation

Welcome to Sun Chili!Soft ASP 3.6.2, a Web server plug-in that enables Web servers running on different platforms to process Active Server Pages (ASP) code.

This documentation provides information about the installation, configuration, and use of Sun Chili!Soft ASP 3.6.2. It also provides basic information about building applications, and reference information.

There are two versions of this documentation: one in HTML format that includes dynamic index and search functionality, and one in Adobe PDF format. To view and print the PDF version, Adobe Acrobat Reader must be installed. To obtain a free copy of Acrobat Reader, go to:

http://www.adobe.com/products/acrobat/readstep2.html

In addition to the documentation, the following resources will also help you learn more about Sun Chili!Soft ASP:

- Diagnostic applications verify that your ASP environment is working correctly.
- Sample ASP applications demonstrate the basics of building Sun Chili!Soft ASP applications.
- The 10-step Tour provides a basic introduction to Sun Chili!Soft ASP technology.

This section describes these resources, and how to access them.

In this section:

We would like to have your comments about this documentation regarding what you found useful, and what could be improved. Please send your comments to us using the feedback form at:

http://www.chilisoft.com/feedback/documentation.asp

What's in This Documentation

This documentation includes information about installing, configuring, and running Sun Chili!Soft ASP. It also introduces the basics of building Sun Chili!Soft ASP applications, and provides reference information about using scripting languages, connecting to databases, and developing and using components.

The documentation is structured so you can easily find the information you need. The following table describes the contents of each chapter, and the Sun Chili!Soft ASP users who will benefit most from reading them.

Chapter Name	Who Should Read It	Description
Introduction: About This Documentation	Everyone	Provides an overview of the Sun Chili!Soft ASP product documentation and other Sun Chili!Soft ASP resources.
Chapter 1: About Sun Chili!Soft ASP	Everyone should read "New in This Release." Users new to ASP technology or Sun Chili!Soft ASP should read the entire chapter.	Describes what is included with this release of Sun Chili!Soft ASP. It also provides an introduction to ASP technology, and describes the Sun Chili!Soft implementation of ASP.
Chapter 2: Installing and Configuring Sun Chili!Soft ASP	System administrators	Explains how to install Sun Chili!Soft ASP on your server, and describes changes the setup program makes to your Web server configuration files. It also provides instructions for performing basic server configuration tasks, and for enabling users to publish ASP applications to the Web server.
Chapter 3: Managing Sun Chili!Soft ASP	System administrators	Provides information about administering Sun Chili!Soft ASP, including information about changing ASP Server configuration settings, configuring security, optimizing server performance, and troubleshooting server problems.
Chapter 4: Building a Sun Chili!Soft ASP Application	Web developers new to ASP technology and Sun Chili!Soft ASP, and system administrators who want a basic introduction to ASP, should read the introductory information about creating a basic ASP application and connecting to a database. Experienced ASP developers might want to read the information about using objects and components. Everyone should read the section about publishing an ASP application.	Introduces the basics of developing ASP applications: creating an ASP page, adding scripts and server-side includes, and defining the application on the server. It also discusses extending ASP applications by using objects and components, and connecting to databases. This chapter concludes with information about publishing a Sun Chili!Soft ASP application.

Chapter 5: Developer's Reference	Web developers	Provides reference information about using built-in ASP components, additional off-the-shelf ASP components, and custom components. It also provides a scripting reference for VBScript and JScript, and reference information about using Active Data Objects (ADO), Chili!Beans, and SpicePack components.
Appendix A: Sun Chili!Soft ASP Error Messages	System administrators and Web developers	Explains error messages you might encounter when using Sun Chili!Soft ASP.
Appendix B: Troubleshooting	System administrators and Web developers	Provides troubleshooting tips for problems you might encounter when running Sun Chili!Soft ASP.
Appendix C: Glossary	System administrators and Web developers	Contains a glossary of terms you might encounter when administering Sun Chili!Soft ASP and developing ASP applications.

Accessing Documentation, Samples, and Diagnostics

The Sun Chili!Soft ASP Start Page provides links to the product documentation, ASP sample applications, diagnostic applications, and the 10-step Tour. You can access the Start Page from:

http://[HOSTNAME]/caspsamp

where [HOSTNAME] is the hostname of your Web server.

To use ASP functionality in the sample applications, diagnostics, and 10-step Tour, **Allow session state** must be set to **yes** on the **Server Settings** page in the Sun Chili!Soft ASP Administration Console. For more information about this setting, see "Enabling Session State" in "Chapter 3: Managing Sun Chili!Soft ASP." For Windows systems, see "Editing the Windows Registry" in "Chapter 3: Managing Sun Chili!Soft ASP."

Product Documentation

On UNIX and Linux systems, product documentation can always be accessed from the Sun Chili!Soft ASP Administration Console. During installation, you have the option to access the documentation from your Web server. If you choose this option and your Web server is running, you can access the HTML version of the documentation at:

http://[HOSTNAME]/caspdoc/

where [HOSTNAME] is the hostname of your Web server.

From the first page of the HTML documentation, you can click a link to open the version in Adobe PDF format. To access the PDF version directly, use the following URL:

http://[HOSTNAME]/caspdoc/pdf/chilisoft_asp_docs.pdf

where [HOSTNAME] is the hostname of your Web server.

You can also access both versions of the documentation from the Sun Chili!Soft Web site at:

http://www.chilisoft.com/caspdoc/

Note

In addition to the complete product documentation, there are two other Sun Chili!Soft ASP documentation resources: the QuickStart Guide and the README file.

The QuickStart Guide provides installation instructions and information about getting started with Sun Chili!Soft ASP (the same information that is provided in "Chapter 2: Installing and Configuring Sun Chili!Soft ASP"). Following installation, the QuickStart Guide can be found in the following directory:

/[C-ASP_INSTALL_DIR]/

where [C-ASP_INSTALL_DIR] is the path name of the Sun Chili!Soft ASP installation directory (/opt/casp by default).

The README file can be accessed as described in "Viewing the Product README File" in "Chapter 3: Managing Sun Chili!Soft ASP."

Sample ASP Applications

The Sun Chili!Soft ASP setup program gives you the option to enable sample ASP applications on the computer running Sun Chili!Soft ASP. If you choose this option and your Web server is running, you can access the samples from the Sun Chili!Soft ASP Start Page at:

http://[HOSTNAME]/caspsamp/

where [HOSTNAME] is the hostname of your Web server.

- or -

You can access the samples from the Sun Chili!Soft Web site at:

http://www.chilisoft.com/caspsamp/

You can view a list of sample ASP applications on the **ASP Applications** page of the Sun Chili!Soft Administration Console, as described in "Adding an ASP Application" in "Chapter 3: Managing Sun Chili!Soft ASP." The sample applications are located in your file system at:

```
/caspsamp="[C-ASP_INSTALL_DIR]/caspsamp"
```

where `/caspsamp` is the directory alias (virtual directory) for the caspsamp sample application and `[C-ASP_INSTALL_DIR]` is the path name of the Sun Chili!Soft ASP installation directory.

In the caspsamp directory, the following subdirectories contain samples:

```
/casp401k="[C-ASP_INSTALL_DIR]/caspsamp/401K/content"
/caspagent="[C-ASP_INSTALL_DIR]/caspsamp/friendship/agent/content"
/caspclient="[C-ASP_INSTALL_DIR]/caspsamp/friendship/client/content"
```

10-step Tour

If you are new to ASP technology, we recommend that you take our 10-step Tour. After installing Sun Chili!Soft ASP and with your Web server running, you can access the tour from the Sun Chili!Soft ASP Start Page at:

http://[HOSTNAME]/caspsamp/

where [HOSTNAME] is the hostname of your Web server.

Diagnostic Applications

The following diagnostic applications are installed with Sun Chili!Soft ASP, and enable you to verify that various features of your ASP environment are working correctly:

- **HELLO** – This application tests the functionality of ASP and VBScript by using a simple "Hello World" script.
- **SERVER** – This application tests the ASP Server-to-Web server connection by retrieving the standard Web server variables.
- **JSCRIPT** – This application tests the functionality of JScript.
- **COMPONENTS** – This application tests the functionality of additional components installed with Sun Chili!Soft ASP.
- **ADO** – This application accesses ActiveX Data Objects (ADO) from VBScript. It connects to a sample database by using ODBC (Open Database Connectivity) to test the functionality of ADO and the dBASE ODBC driver.
- **JSADO** – This application performs the same test as the ADO diagnostic, except it accesses ADO from JScript rather than VBScript.
- **SQLEXECUTE** – This application uses ADO to execute an SQL statement and display the results. To use this application, you must first create a system DSN for your database on the ASP Server, as described in "Adding a DSN" in "Chapter 3: Managing Sun Chili!Soft ASP."

When your Web server is running, you can access the diagnostic applications from the Sun Chili!Soft ASP Start Page at:

http://[HOSTNAME]/caspsamp/

where [HOSTNAME] is the hostname of your Web server.

Chapter 1: About Sun Chili!Soft ASP

Sun Chili!Soft ASP enables you to run ASP applications on a variety of Web servers running under Sun Solaris, Sun Cobalt, Linux, IBM AIX, Hewlett-Packard HP-UX, and Microsoft Windows NT and Windows 2000 operating systems.

This chapter describes what's new in this release of Sun Chili!Soft ASP. It also provides an overview of Active Server Pages (ASP) technology, and describes the Sun implementation of ASP.

Who should read this chapter: Everyone should read the first topic, "New in This Release." Users new to ASP or Sun Chili!Soft ASP should read the entire chapter.

In this chapter:

New in This Release

Sun Chili!Soft ASP 3.6.2 includes the following new features:

UNIX and Linux

- **iPlanet Web Server 6.0 support and detection:** Sun Chili!Soft ASP for Solaris, HP-UX, AIX, and Linux now provides support for iPlanet Web Server 6.0. Sun Chili!Soft ASP 3.6.2 is available free for iPlanet Web Server 6.0. When iPlanet Web Server 6.0 is detected, you automatically receive a full, unlimited license.
- **Apache Web Server 1.3.22 and Zeus Web Server 4.0 support:** Sun Chili!Soft ASP for Solaris and Linux now provides support for Apache 1.3.22 and Zeus 4.0.
- **DataDirect Connect ODBC 4.0 (Wire Protocol drivers) installation:** Sun Chili!Soft ASP 3.6.2 for UNIX and Linux installs ODBC drivers for many different databases, and includes the DataDirect Connect ODBC 4.0 drivers (DataDirect is a former business unit of MERANT). For information about the ODBC drivers included in Sun Chili!Soft ASP 3.6.2, see the installation requirements section for your specific platform in "Installing and Uninstalling Sun Chili!Soft ASP" in "Chapter 2: Installing and Configuring Sun Chili!Soft ASP." For information about configuring the drivers for the data source being used, see "Configuring a Database" in "Chapter 3: Managing Sun Chili!Soft ASP."
- **Installation changes:** Sun Chili!Soft ASP 3.6.2 for UNIX and Linux now has just one installation process, which combines the ease and flexibility of the Bundle and Custom installation options provided in previous versions. Sun Chili!Soft ASP 3.6.2 includes a

ready-to-run Apache 1.3.19 Web server, Sun Chili!Beans support, and Sun SpicePack components. For installation information, see "Installing and Uninstalling Sun Chili!Soft ASP" in "Chapter 2: Installing and Configuring Sun Chili!Soft ASP."

- **Sun SpicePack installation:** Sun SpicePack components (Chili!Mail, Chili!POP3, and Chili!Upload) are now included and installed with Sun Chili!Soft ASP 3.6.2 for UNIX and Linux. You do not need to purchase the SpicePack separately. For SpicePack information, see "SpicePack Component Reference" in "Chapter 5: Developer's Reference."
- **Product Updates:** Sun Chili!Soft ASP 3.6.2 now includes a mechanism that notifies you of product updates, and enables you to quickly and easily obtain them. For update information, see "Checking for Product Updates" in "Chapter 3: Managing Sun Chili!Soft ASP."
- **Microsoft FrontPage 2002 Server Extensions support:** Sun Chili!Soft ASP 3.6.2 supports, but does not install FrontPage 2002 Server Extensions. For FrontPage publishing information, see "Enabling FrontPage Publishing" in "Chapter 3: Managing Sun Chili!Soft ASP."

Microsoft Windows NT and Windows 2000

- **iPlanet Web Server 6.0 detection:** Sun Chili!Soft ASP 3.6.2 is available free for iPlanet Web Server 6.0. When iPlanet Web Server 6.0 is detected, you automatically receive a full, unlimited license.
- **iPlanet Web Server 6.0 and Apache Web Server 1.3.22 support:** Sun Chili!Soft ASP for Windows now provides support for iPlanet 6.0 and Apache 1.3.22.

For detailed installation and application notes, see "Installing and Uninstalling Sun Chili!Soft ASP" in "Chapter 2: Installing and Configuring Sun Chili!Soft ASP." Important information is also provided in the README file that was installed with your software. You can access the README as described in "Viewing the Product README File" in "Chapter 3: Managing Sun Chili!Soft ASP."

Supported Platforms and Web Servers

Sun Chili!Soft ASP 3.6.2 supports the following operating systems and Web servers:

Version	Operating System	Web Servers
Sun Chili!Soft ASP 3.6.2 for Solaris	Sun Solaris 2.6, 7, and 8	Apache 1.3.19 DSO Apache 1.3.22 DSO iPlanet Web Server, Enterprise Edition 6.0 SP1 Zeus Web Server 4.0
Sun Chili!Soft ASP 3.6.2 for HP-UX	HP-UX 11.0	iPlanet Web Server, Enterprise Edition 6.0 SP1

		HP Apache-based Web Server 1.3.19.23
		Apache 1.3.19 DSO
Sun Chili!Soft ASP 3.6.2 for Linux	Red Hat Linux 7.2	Apache 1.3.19 DSO
	SuSE 7.3 Professional	Apache 1.3.22 DSO
	Mandrake Linux 8.1	iPlanet Web Server, Enterprise Edition 6.0 SP1
	Debian 2.2r5	
		Zeus Web Server 4.0
Sun Chili!Soft ASP 3.6.2 for Windows	Microsoft Windows NT Server 4.0 SP6	iPlanet Web Server, Enterprise Edition 6.0 SP1
	Microsoft Windows 2000 Server SP1	Apache 1.3.22

Note

Sun Chili!Soft ASP 3.6.2 may install to other versions of the supported Web servers listed above. However, versions not listed have not been certified to run with Sun Chili!Soft ASP 3.6.2, and their use is not supported by Sun Chili!Soft Customer Support.

For information about supported database types, see the installation requirements section for your specific platform in "Installing and Uninstalling Sun Chili!Soft ASP" in "Chapter 2: Installing and Configuring Sun Chili!Soft ASP."

What is ASP?

ASP is a server-side scripting technology developed by Microsoft. It is an open, compile-free application environment in which you can combine HTML, scripts, and reusable components to build dynamic and powerful Web applications.

An ASP application consists of ASP pages published on a Web site. ASP pages can contain HTML code, client-side scripts, and server-side scripts. When a user requests an ASP page, the Web server calls the ASP Server, which processes the requested file from top to bottom, executing any server-side scripts. It then formats a standard Web page and sends the results to the user's browser.

Because scripts can run on the server rather than on the client, the Web server can do much of the work involved in generating the HTML pages sent to browsers. Server-side scripts cannot be readily copied because only the result of the script is returned to the browser. Users cannot view the script commands that created the page they are viewing.

ASP was designed as a faster and easier alternative to Common Gateway Interface (CGI) scripting using Perl or C scripts. ASP provides an easy-to-learn scripting interface (including native support for both VBScript and JScript), along with a number of predefined objects that simplify many development tasks, such as maintaining user state and defining global variables within an application. You can also use Active-X Data Objects (ADO) components to perform

additional functions, including accessing ODBC-compliant databases and outputting data to text files.

You can extend ASP scripts by using Java components and extensible markup language (XML) .

ASP runs as a service of the Web server, and is optimized for multiple threads and multiple users. This means that ASP is fast and easy to implement. ASP enables you to separate the design of your Web page from the details of programming access to databases and applications, so programmers and Web designers can focus exclusively on what they do best.

Following are a few examples of what you can do with ASP applications. You can:

- Put your employee handbook online, and build an application that allows employees to update their information.
- Connect customer orders from an online storefront to an existing inventory database and order-processing system.
- Give visitors a personalized view of information on your Web site, flagging items that are new since their last visit.

See also:

- ASP for the HTML Author in this chapter
- ASP for the Experienced Web Scripter in this chapter
- ASP for the Web Developer and Programmer in this chapter
- Chapter 4: Building a Sun Chili!Soft ASP Application

ASP for the HTML Author

For the HTML author, ASP is an easy way to begin creating Web applications. To process user input on the Web server with Common Gateway Interface (CGI) applications, you must learn a programming language such as Perl or C. With ASP, however, you can collect HTML form information and pass it to a database by using simple server-side scripts written in VBScript or Jscript that are embedded directly in your HTML documents. You can use server-side ASP scripts to store HTML form information in a database, personalize Web sites according to visitor preferences, or use different HTML features based on the browser.

See also:

- What is ASP? in this chapter
- Chapter 4: Building a Sun Chili!Soft ASP Application

ASP for the Experienced Web Scripter

ASP is language-neutral, so if you are skilled at a scripting language such as VBScript or JScript, you already know how to use ASP. (Sun Chili!Soft ASP comes with VBScript and JScript scripting engines.) You can use server-side ASP scripts to store HTML form information in a

database, personalize Web sites according to visitor preferences, or use different HTML features based on the browser.

See also:

- What is ASP? in this chapter
- Chapter 4: Building a Sun Chili!Soft ASP Application

ASP for the Web Developer and Programmer

If you develop Web applications by using a programming language such as Visual Basic, C++, or Java, you will appreciate the flexibility of ASP. Besides using scripts to create an engaging HTML interface for your application, you can also use Java components to encapsulate your application's business logic into reusable modules that can be called from a script, from another component, or from another program.

See also:

- What is ASP? in this chapter
- Chapter 4: Building a Sun Chili!Soft ASP Application

What is Sun Chili!Soft ASP?

Sun Chili!Soft ASP 3.6.2 is a platform-independent implementation of Microsoft Active Server Pages (ASP) technology. It is the functional and syntactic equivalent of the ASP technology included with Microsoft Internet Information Server (IIS). Unlike the Microsoft implementation, however, Sun Chili!Soft ASP enables you to build ASP applications that run on many different systems, in addition to Windows and IIS. You can build powerful ASP applications that run on popular Web servers under Sun Solaris, Sun Cobalt, Linux, IBM AIX, Hewlett-Packard HP-UX, and Windows NT and Windows 2000. Sun Chili!Beans support even enables you to use your Java objects with ASP applications.

See also:

- What is ASP? in this chapter
- Sun Chili!Soft ASP Features in this chapter
- Chapter 4: Building a Sun Chili!Soft ASP Application

Sun Chili!Soft ASP Features

Sun Chili!Soft ASP is designed for the easy setup, administration, and deployment of Active Server Pages on Solaris, Linux, AIX, HP-UX, and Windows NT and Windows 2000. Sun Chili!Soft ASP includes the following key features:

- **One-hundred percent pure Active Server Pages support:** Sun Chili!Soft ASP supports the most common and frequently used ASP standards, such as ASP 2.0, VBScript 3.2, and JScript 3.2. You can easily deploy new or existing ASP applications with few or no changes to code.
- **Authentic Microsoft scripting engines:** Sun Chili!Soft ASP includes authentic Microsoft VBScript and JScript scripting engines, guaranteeing complete code compatibility and preventing extended debugging cycles for ASP applications.
- **Database connectivity tools suite:** Sun Chili!Soft ASP includes support for ADO 2.0 and provides ODBC drivers for all major databases (UNIX and Linux versions). ODBC drivers are provided by DataDirect Technologies (a former business unit of MERANT), the leading provider of enterprise database connectivity. With Sun Chili!Soft ASP, you have everything you need to integrate data with your ASP application. Drivers do not need to be purchased separately.
- **Chili!Beans COM-to-Java bridge:** Because COM (Component Object Model) objects have limited portability, the use of Java components is strongly recommended. Sun Chili!Beans provides full access to Java's object-oriented environment directly from ASP scripting, and acts as a dynamic COM wrapper that exposes the public methods and properties of the wrapped Java class. This functionality allows you to integrate these components into your ASP Web applications. Chili!Beans is the fastest, most efficient method of developing ASP applications that are both portable and extendable.
- **Powerful and scalable ASP processing:** Sun Chili!Soft ASP includes a browser-based Administration Console for easy management and configuration of the ASP Server engine. It also provides server monitoring and diagnostic tools for real-time tracking and monitoring of server performance and availability, and ASP error logging for quick detection and diagnosis of server errors.
- **Web application developer tool integration:** Sun Chili!Soft ASP works directly with leading Web application development tool vendors such as Macromedia, Adobe, and Microsoft to ensure that your applications work seamlessly with Sun Chili!Soft ASP.
- **World-class support:** Sun Chili!Soft ASP is backed by Sun's global support network and its world-class support.

See also:

- What is ASP? in this chapter
- What is Sun Chili!Soft ASP? in this chapter
- Chapter 4: Building a Sun Chili!Soft ASP Application
- Supported Platforms and Web Servers in "Chapter 1: About Sun Chili!Soft ASP"

Chapter 2: Installing and Configuring Sun Chili!Soft ASP

This chapter describes the basic steps involved with installing Sun Chili!Soft ASP on your server, and with enabling users to publish ASP applications. This chapter also tells you how to configure the server for hosting ASP applications and for connecting ASP applications to databases. "Chapter 3: Managing Sun Chili!Soft ASP" provides detailed information about such topics as changing server configuration settings, configuring different types of databases, and optimizing server performance.

Who should read this chapter: System administrators responsible for installing, configuring, and running Sun Chili!Soft ASP.

In this chapter:

- Installing and Uninstalling Sun Chili!Soft ASP
- Enabling Publishing
- Defining ASP Applications on the Server
- Enabling Database Connections on the Server

Installing and Uninstalling Sun Chili!Soft ASP

This section includes the following topics about installing and uninstalling Sun Chili!Soft ASP:

- Installing Sun Chili!Soft ASP for Sun Solaris
- Installing Sun Chili!Soft ASP for HP-UX
- Installing Sun Chili!Soft ASP for Linux
- Installing Sun Chili!Soft ASP for Windows
- Configuring the Web Server after Installation
- Changes to Web Server Configuration Files
- Changing Installation Options after Installation
- Uninstalling Sun Chili!Soft ASP

Installing Sun Chili!Soft ASP for Solaris

This section describes the requirements and procedures for installing Sun Chili!Soft ASP 3.6.2 for Sun Solaris. For the most up-to-date information about this product, see the README file included with your software. You can access the README file from the Sun Chili!Soft ASP

Administration Console, as described in "Viewing the Product README File" in "Chapter 3: Managing Sun Chili!Soft ASP."

In this section:

- Installation Requirements: Sun Chili!Soft ASP for Solaris
- Running the Setup Program: Sun Chili!Soft ASP for Solaris
- Important Notes about Solaris Installations
- Upgrading Sun Chili!Soft ASP for Solaris

Installation Requirements: Sun Chili!Soft ASP for Solaris

This section lists the hardware and software requirements for installing and running Sun Chili!Soft ASP 3.6.2 for Solaris. It also lists the additional software (such as databases and tools) that Sun Chili!Soft ASP supports.

Hardware and Software Requirements

Hardware

Your hardware configuration must meet the following minimum requirements:

- 256 MB RAM
- SPARC processor
- 140 MB free hard disk space

Operating System

Sun Chili!Soft ASP for Solaris runs on the following operating systems:

- Sun Solaris 2.6, 7, and 8

Web Server

You must be running one of the following Web servers:

- Apache 1.3.19 DSO
- Apache 1.3.22 DSO
- iPlanet Web Server, Enterprise Edition 6.0 SP1
- Zeus Web Server 4.0

Note

Sun Chili!Soft ASP 3.6.2 may install to other versions of the supported Web servers listed above (Apache 1.3.12, for example). However, versions not listed have not been certified to run with Sun Chili!Soft ASP 3.6.2, and their use is not supported by Sun Chili!Soft Customer Support.

Supported Software

This section lists additional software that is supported and/or installed by Sun Chili!Soft ASP.

Database

Sun Chili!Soft ASP for Solaris includes ODBC drivers for the following databases:

DataDirect Connect ODBC 4.0 (Wire Protocol drivers)

- DB2 Universal Database (UDB) v7.1
- dBASE 5
- Informix Dynamic Server 9.x
- Informix Dynamic Server 2000 (9.20)
- Microsoft SQL Server 7.0 SP1
- Microsoft SQL Server 2000 SP1
- Oracle 8i (8.1.6)
- Oracle 8i (8.1.7)
- Oracle 9i
- Sybase Adaptive Server Enterprise 11.9.2
- Sybase Adaptive Server Enterprise 12.5
- Text Files

DataDirect SequeLink 4.5.1a

- Microsoft SQL Server 6.5 (via SequeLink)
- Microsoft Access 2000, 97, and 95 (via SequeLink)

Open Source

- MySQL 3.22.30 and 3.23.49
- PostgreSQL 6.5.2 and 7.1.3

ODBC drivers are installed automatically by the Sun Chili!Soft ASP setup program. Following installation, you must configure the drivers for the data source being used. For more information, see "Configuring a Database" in "Chapter 3: Managing Sun Chili!Soft ASP."

Note

To provide remote database connectivity with Microsoft Access and Microsoft SQL Server 6.5 databases, Sun Chili!Soft ASP ships with the client portion of the DataDirect SequeLink 4.51a software. On the Windows machine that contains your database, you also must download and install the server portion of this software from Sun Chili!Soft. For more information, see "Configuring SequeLink" in "Chapter 3: Managing Sun Chili!Soft ASP."

Sun Chili!Soft ASP 3.6.2 may work with other versions of the databases listed above. However, versions not listed have not been tested to run with Sun Chili!Soft ASP 3.6.2, and their use is not supported by Sun Chili!Soft Customer Support.

Java Runtime Environment

Sun Chili!Soft ASP 3.6.2 includes the Java Runtime Environment (JRE) 1.3.1 for Sun Chili!Beans. While JREs 1.2.x and 1.3.x are supported, and the Java 2 Runtime Environment can also be used, the use of JRE 1.3.1 is strongly recommended. Chili!Beans enables you to use Java class files as components called from VBScript or JScript. To use Chili!Beans, a Java runtime environment must be installed on the machine, and Chili!Beans must be enabled from the Administration Console. For more information, see "Chili!Beans Component Reference" in "Chapter 5: Developer's Reference."

Microsoft FrontPage Server Extensions

Sun Chili!Soft ASP enables you to run ASP pages generated by Microsoft FrontPage. Microsoft FrontPage Server Extensions provide services to the Web server that work in conjunction with Sun Chili!Soft ASP to provide FrontPage functionality to computers not running Microsoft Windows NT or Windows 2000 and Internet Information Services (IIS).

FrontPage Server Extensions are no longer installed with Sun Chili!Soft ASP; you must obtain them from Microsoft. See "Enabling FrontPage Publishing" in "Chapter 3: Managing Sun Chili!Soft ASP."

Note

> Specific questions about the installation, configuration, and use of FrontPage and FrontPage Server Extensions should be directed to Microsoft or its representatives.

See also:

Installing Sun Chili!Soft ASP for Solaris in this chapter

Running the Setup Program: Sun Chili!Soft ASP for Solaris

The Sun Chili!Soft ASP setup program takes you through the steps required to install Sun Chili!Soft ASP 3.6.2 on Sun Solaris. The setup program installs the following components on the target server:

- Sun Chili!Soft ASP Server
- Sun Chili!Soft ASP Administration Console
- Sun Chili!Beans support (the use of Chili!Beans is optional). Sun Chili!Soft ASP 3.6.2 includes Java Runtime Environment (JRE) 1.3.1. The use of this specific JRE version is not required, but is strongly recommended.
- Sun SpicePack components
- Apache Web Server 1.3.19 DSO (preconfigured or "bundled" with Sun Chili!Soft ASP 3.6.2).

The setup program gives you the option to either specify the configuration options or accept the default configuration settings. The configuration settings you specify during installation can have

serious consequences for your server environment. Before you begin, make sure you meet the following requirements:

- You are logged in as root.
- You have 60 MB of hard disk space available to extract the installation files.
- Your hardware and software meet the requirements listed in "Installation Requirements: Sun Chili!Soft ASP for Solaris" in this chapter.
- You know your product serial number. If you do not know your product serial number and you go ahead with the installation, you will receive an evaluation license.
- You know which language you want Sun Chili!Soft ASP to support (English - US, Japanese Shift-JIS, and so on) . If desired, you can change the language after installation, as described in "Configuring International Support" in "Chapter 3: Managing Sun Chili!Soft ASP."
- To use Chili!Beans, a Java runtime environment (JRE) must be installed on the machine, and Chili!Beans must be enabled from the Administration Console. During installation, Sun Chili!Soft ASP will install JRE 1.3.1 unless you choose to install another supported version (1.2.x or 1.3.x). Note that while you have the option to install a different JRE, and the Java 2 Runtime Environment can also be used, the use of JRE 1.3.1 is strongly recommended. Chili!Beans enables you to access Java objects and classes from an ASP script. For more information, see "Chili!Beans Component Reference" in "Chapter 5: Developer's Reference."
- If you are upgrading from a previous version of Sun Chili!Soft ASP, you must install Sun Chili!Soft ASP 3.6.2 in the same directory as your previous installation. Also, be sure to review the information in "Upgrading Sun Chili!Soft ASP for Solaris" in this chapter.
- If you choose to customize settings for the Sun Chili!Soft ASP Administration Console, you must specify a username and password for accessing the console. Be sure to specify a password that you can remember, or else plan to store the password in a secure location. If you forget your password, you can run the admtool utility and set a new one, as described in "Configuring Usernames and Passwords" in "Chapter 3: Managing Sun Chili!Soft ASP." If you do not choose the customize settings option, the username is set as "admin" and the password as "root." To protect the security of your server, you should run the admtool and change these as soon as possible following installation.
- If you are installing Sun Chili!Soft ASP to a Zeus Web server, you must take additional steps prior to installation. See "Installing Sun Chili!Soft ASP to a Zeus Web Server" immediately below. If you are not installing to a Zeus Web server, proceed with the installation as described in "To run the setup program" in this section.

Installing Sun Chili!Soft ASP to a Zeus Web Server

Before you can install Sun Chili!Soft ASP to a Zeus Web server, you must first enable NSAPI (Netscape’s Web server API) for a Zeus Virtual Server, and then copy the NSAPI configuration files to the proper location. For detailed information beyond what is provided here, see your Zeus product documentation.

Note

When configuring the Zeus Web server to run with Sun Chili!Soft ASP 3.6.2, do not specify the Web server hostname as the fully qualified domain name (or "server address"). If you do, the Test DSN feature of the Administration Console will not work. Instead, use the hostname only. For example, do not use: hostname.domain.com. Instead, use: hostname.

To enable NSAPI for a Zeus Virtual Server

1. Go to the **Configuration Summary** page for the Zeus Virtual Server.
2. Go to the **Enabling NSAPI Functionality** page by clicking the **NSAPI** link.
3. Click the **Enabled** button, write down the value of `[path_prefix]` for future reference, and then click the **Apply changes** button.
4. Commit the changes by clicking the yellow sun graphic in the top right section of the page, and then click the **Commit** button.

   NSAPI has been enabled for a Zeus Virtual Server. You must now copy the NSAPI configuration files, as described in the following procedure.

To copy the NSAPI configuration files

Zeus provides copies of the NSAPI configuration files, which are typically located in the following directory:

```
[zeushome]/webadmin-4.0r1/docroot/nsapi-skel/
```

where [zeushome] is the root of your Zeus installation (that is, /usr/local/zeus).

Once the files have been located, use the following commands:

```
#> mkdir –p [path_prefix]/https-[virtual_server_name]/config
#> cd [path_prefix]/https-[virtual_server_name]/config
#> cp –r [zeushome]/webadmin-4.0r1/docroot/nsapi-skel/* .
```

where `[path_prefix]` is the value you wrote down in step 3 of the previous procedure (the default will be `[zeushome]/ns-config`), and `[virtual_server_name]` is the name of the Zeus Virtual Server. Zeus is now configured correctly for the installation of Sun Chili!Soft ASP.

To run the setup program and install Sun Chili!Soft ASP, use the following procedure.

To run the setup program

1. If you are installing Sun Chili!Soft ASP 3.6.2 from the CD-ROM, go to step 2. If you are downloading Sun Chili!Soft ASP 3.6.2 from the Web, download the casp-3.6.2-solaris.tar file to a temporary directory, and then extract the installation files by using the following command:

   ```
   #> tar -xvf casp-3.6.2-solaris.tar
   ```

This step creates the following files in the temporary directory, which you can delete after completing the installation:

A QuickStart Guide with installation instructions
A README file containing important product information
The install.sh installation script
A package/EULA file containing the End-User License Agreement
A package/directory containing files required to install Sun Chili!Soft ASP

2. Run the install.sh script by using the following command. For downloaded versions, this script is located in the temporary directory you created in the previous step. For CD-ROM versions, this script is located on your installation CD-ROM.

```
#> ./install.sh
```

3. Review the End-User License Agreement (EULA) that displays. Enter **yes** if you agree with its conditions. You must type the entire word, "yes."

   – or –

   Enter **no** if you do not agree. If you enter **no**, the Sun Chili!Soft ASP setup program quits the installation.

4. On the next screen, press **Enter** to accept the default Sun Chili!Soft ASP installation directory (/opt/casp).

   – or –

   Enter the absolute path name of a different installation directory. Make note of this location, so you can easily find the Sun Chili!Soft ASP files at a later time.

   Note: To upgrade from the previous version, you must specify the installation directory of your existing installation.

5. If the setup program detects a previous installation of Sun Chili!Soft ASP in the specified directory, you will be prompted to uninstall the previous version (if a previous installation is not detected, go to step 6):

   Enter **n** (no) to change the installation directory or cancel the installation.

   – or –

   Enter **y** (yes) to uninstall the previous installation. You will then be prompted to proceed. At this confirmation prompt, enter **n** to cancel the installation, or **y** to uninstall the previous version. If you enter **y**, the previous version will be uninstalled by the installer. Once that process is finished, press **Enter** to continue.

   Note: At the end of the Sun Chili!Soft ASP installation process you will be prompted to import your settings.

6. On the next screen, enter **y** (yes) if you know the product serial number.

   – or –

   If you do not know the serial number, enter **n**. An evaluation license will be installed (go to step 8).

Note: iPlanet 6.0 users are already licensed to use this product and do not need to enter a serial number. If you are using iPlanet 6.0, enter **n** and you will receive a full, unlimited license (go to step 8).

7. When prompted, enter the product serial number.
8. Sun Chili!Soft ASP 3.6.2 includes a ready-to-run Apache 1.3.19 Web server configured with support for Microsoft FrontPage 2002 Server Extensions (the extensions themselves are not installed) and EAPI (Extended API). If you have not yet configured a Web server, you have the option to install this preconfigured Apache Web server.

   Enter **y** (yes) to install the preconfigured ("bundled") Apache 1.3.19 Web server, and proceed to the next step.

   – or –

   Enter **n** (no) if you do not want to install the preconfigured Apache Web server, and proceed to step 10.

   Note: Installing the preconfigured Apache Web server will not affect any other installed Web server. However, make sure you do not install the preconfigured Apache Web server to the same port as that of an existing Web server.
9. The Apache Web server has many settings that you can select and configure. Specify the desired configuration option for the preconfigured Apache 1.3.19 Web server:

   Enter **1** (**Default configuration**) to direct Sun Chili!Soft ASP to automatically configure all settings for you.

   – or –

   Enter **2** (**Specify only the Web server listen port**) to specify only the port number that the Apache Web server will listen on, and then specify the port.

   – or –

   Enter **3** (**Customize the configuration**) to specify the configuration of many different settings, and then respond to the prompts.

   Note: All Apache configuration settings can be changed manually after installation by editing the Apache configuration file.
10. On the next screen, press **Enter** to accept the default language (English – US).

    – or –

    Enter the number of a language shown in the list.
11. On the next screen, enable or disable Chili!Beans support as desired:

    Select **Use the bundled 1.3.1 JRE** to enable Chili!Beans support and install JRE 1.3.1 (this is the recommended JRE), and then respond to the prompts.

    – or –

Select **Specify the path to an existing JRE** to specify the path to an existing JRE (JRE versions 1.2.x and 1.3.x are supported), or type **none** to return to the previous screen, and then respond to the prompts.

– or –

Select **Disable Java support** to disable Chili!Beans support, and then respond to the prompts.

– or –

Select **Keep your current settings**.

12. The next screen provides options for choosing the Web server to configure with Sun Chili!Soft ASP. If you want the setup program to search your system and generate a list of installed Web servers from which to choose, enter the number of the desired search option (**1**, **2**, or **3**). If you select one of these search options, skip to step 15.

    – or –

    If you want to skip this search and provide information about the Web server, enter **4** (**Don't search**). The installer will not search for the Web server, and instead takes you to step 13.

    Note: If you chose to install the preconfigured ("bundled") Apache 1.3.19 Web server in step 8, go to step 15.

13. On the next screen, enter the number of the type of Web server to configure: **1** for Apache, **2** for iPlanet, or **3** for Zeus, and go to step 14.

    – or –

    To cancel the user-specified Web server configuration, enter **4**. If you choose this option, the setup program returns you to the previous screen. From this screen you can choose another option, as described in step 15.

14. At the prompts, enter the path of your Web server. If you do not know the correct path, type **none** to cancel (this takes you to step 15). If you selected Apache in step 13, you will be prompted for the Web server binary. After all information has been added correctly, the installer takes you to step 15. There you will be able to select the Web server that was just added.

15. On the next screen, enter the number of the Web server you want the setup program to configure to run with Sun Chili!Soft ASP, and go to step 16 (if you selected option **1**, **2**, or **3** in step 12).

    – or –

    To provide information about the Web server to configure, enter the number for the option **Specify the Web server**. If you choose this option, follow the instructions in step 13.

    – or –

    To perform another search for installed Web servers, enter the number for the option **Attempt to auto-detect more Web servers**. If you choose this option, follow the instructions in step 12.

– or –

If you do not want to configure a Web server during installation, enter the number for the option **Do not configure a Web server**, and then go to step 18. If you choose this option, the first time you open the Sun Chili!Soft ASP Administration Console you will be prompted to configure the ASP Server and a Web server.

16. On the next screen, verify the information that the setup program displays about your Web server. If the information is incorrect, the Sun Chili!Soft ASP installation could fail. If the installation does fail, you can run the installation script again. If you do this, it is recommended that you first run the uninstall script, as described in "Uninstalling Sun Chili!Soft ASP for UNIX and Linux" in this chapter. If the information on the screen is correct, enter **y** (yes), and go to step 17.

    – or –

    If the information is incorrect, enter **n** (no). The setup program returns you to the previous screen (step 15).

17. On the next screen, choose a configuration option for the ASP Server:

    Enter **1** to choose a default configuration, and go to step 18.

    – or –

    Enter **2** to choose a custom configuration, and enter the following information at the prompts:

    Enable samples on this Web server? Enter **y** (yes) if you want the setup program to install and configure the Sun Chili!Soft ASP samples on your Web server, or enter **n** (no).

    Enable documentation on this Web server? Enter **y** (yes) if you want the setup program to enable the Sun Chili!Soft ASP documentation on your Web server, or enter **n** (no). If you enter **n**, the documentation will be available from the Administration Console, but not from the Sun Chili!Soft ASP Start Page.

    Start the ASP Server on system boot? Enter **y** (yes) if you want the setup program to start Sun Chili!Soft ASP automatically each time you start the computer, or enter **n** (no).

    Would you like this [Web server] restarted? Enter **y** (yes) if you want the setup program to automatically restart the Web server after completing Sun Chili!Soft ASP installation, or enter **n** (no).

    – or –

    Enter **3** to choose another Web server to configure for Sun Chili!Soft ASP, and follow the instructions in step 12.

18. On the next screen, select a configuration option for the Administration Console. When you choose **Default configuration**, the setup program configures the console using default settings. With this option, the username is specified as "admin" and the password as "root." To protect the security of your server, it is strongly recommended that you change these settings as soon as possible. For information about doing this, see "Configuring Usernames and Passwords" in "Chapter 3: Managing Sun Chili!Soft ASP."

If you choose **Custom configuration**, enter the following information at the prompts:

Administration Console port number: Enter the number of the port on which the Web server should listen for requests for the Administration Console, or press **Enter** to accept the default.

Automatically start the Administration Console on system startup: Enter **y** (yes) if you want the Sun Chili!Soft ASP Administration Console to start automatically each time you start the computer, or enter **n** (no).

Type the username…: Enter the administrator username.

New password: Enter the administrator password, which is required to access the Administration Console. Make note of this password, because you cannot access the Administration Console without providing it.

Confirm password: Reenter the password.

19. The next screen provides summary information about your Administration Console configuration. Make note of this information or else print this screen for future reference. When finished, press **Enter** to complete the installation.
20. If you are upgrading from the previous version for Solaris and the setup program was able to export the settings from your previous installation, you will be prompted to import these settings into your new installation. Enter **y** (yes) to import your settings.

    – or –

    Enter **n** (no) if you do not want to import your settings.

The setup program then completes the installation and writes a summary file to:

[C-ASP_INSTALL_DIR]/logs/install_summary

where [C-ASP_INSTALL_DIR] is the directory in which you installed Sun Chili!Soft ASP (/opt/casp by default). Be sure to print the information contained in the install_summary file for future reference.

Before running your installation of Sun Chili!Soft ASP 3.6.2, see "Important Notes about Solaris Installations" in this chapter. If you are upgrading from a previous version of Sun Chili!Soft ASP for Sun Solaris, see "Upgrading Sun Chili!Soft ASP for Solaris" in this chapter.

For information about changes the setup program makes to your Web server configuration files, see "Changes to Web Server Configuration Files" in this chapter. For information about how to change some of the options that were configured during installation, see "Changing Installation Options after Installation" in this chapter.

See also:

Uninstalling Sun Chili!Soft ASP for UNIX and Linux in this chapter

Important Notes about Solaris Installations

Before running your installation of Sun Chili!Soft ASP 3.6.2 for Sun Solaris, review the following information:

- If you chose the default configuration for the Administration Console, the username is configured as "admin" and the password as "root." To protect the security of your server, it is strongly recommended that you change the username and password as soon as possible. For more information, see "Configuring Usernames and Passwords" in "Chapter 3: Managing Sun Chili!Soft ASP."
- You use the Administration Console to start and stop the Sun Chili!Soft ASP Server and to change configuration settings. You can access the Administration Console by typing the following URL in your Web browser address bar:

  http://[HOSTNAME]:[PORT]

  where [HOSTNAME] is the valid DNS name or IP address of the Apache Web server running the Sun Chili!Soft ASP Administration Console and [PORT] is the port on which the Administration Console is configured to run (5100 by default). The URL for accessing the Administration Console is provided in the following file:

  [C-ASP_INSTALL_DIR]/logs/install_summary

  where [C-ASP_INSTALL_DIR] is the directory in which you installed Sun Chili!Soft ASP (/opt/casp by default).
- If you chose not to configure a Web server to run with Sun Chili!Soft ASP during installation, the first time you open the Administration Console you will be prompted to install a Web server.
- The Sun Chili!Soft ASP setup program makes certain changes to the configuration files for the associated Web server. For more information, see "Changes to Web Server Configuration Files" in this chapter.
- Following installation, you can change some of the options that were configured during installation, such as the Web server with which Sun Chili!Soft ASP is configured to run, and the status of Java support (enabled or disabled). For more information, see "Changing Installation Options after Installation" in this chapter.
- To use Chili!Beans, a Java runtime environment (JRE) must be installed on the machine, and Chili!Beans must be enabled from the Administration Console (JRE 1.3.1 is included with Sun Chili!Soft ASP 3.6.2 and is the recommended version). If you did not install a JRE during the installation of Sun Chili!Soft ASP, you can do so by following the instructions in "Enabling Java Support" in this chapter.
- When finished installing Sun Chili!Soft ASP 3.6.2, use the diagnostic applications to verify that your installation is functioning correctly. You can access the diagnostics from the following URL:

  http://[HOSTNAME]/caspsamp/diagnostics.htm

  where [HOSTNAME] is the hostname of the Web server configured to run with Sun Chili!Soft ASP.

- Important summary information about the installation is located in the following file:

  [C-ASP_INSTALL_DIR]/logs/install_summary

  where [C-ASP_INSTALL_DIR] is the directory in which you installed Sun Chili!Soft ASP (/opt/casp by default).

  Be sure to print this information for future reference.

- Certain Sun Chili!Soft ASP 3.6.2 settings have important implications for the security of the Sun Chili!Soft ASP Server. See "Securing the Server" in "Chapter 3: Managing Sun Chili!Soft ASP" to ensure that the settings are appropriately configured for your specific environment.

See also:

Uninstalling Sun Chili!Soft ASP for UNIX and Linux in this chapter

Upgrading Sun Chili!Soft ASP for Solaris

Sun Chili!Soft ASP 3.6.2 automatically upgrades many of your settings from an existing installation of Sun Chili!Soft ASP for Solaris.

To perform an upgrade from Sun Chili!Soft ASP 3.6.0-P2 or 3.6.1 to Sun Chili!Soft ASP 3.6.2, follow the instructions in "Running the Setup Program: Sun Chili!Soft ASP for Solaris" in this chapter. In addition, review the following information and take any necessary steps:

- The Sun Chili!Soft ASP 3.6.2 setup program preserves the settings from your Sun Chili!Soft ASP 3.6.0-P2 or 3.6.1 installation, and then imports these settings into your new Sun Chili!Soft ASP 3.6.2 installation. The settings imported from your existing installation will override the settings that you configure during installation, with the exception of Chili!Beans support. During installation, you must configure Chili!Beans support and select the option for the bundled JRE 1.3.1 (or specify a path to a different JRE). While JREs 1.2.x and 1.3.x are supported, and the Java 2 Runtime Environment can also be used, the use of JRE 1.3.1 is strongly recommended. Java support must be configured, even if it was enabled in the installation you are upgrading.

Note

SpicePack settings and Sybase DSNs are not migrated when upgrading to Sun Chili!Soft ASP 3.6.2.

- If necessary, upgrade your Web server to meet the requirements listed in "Installation Requirements: Sun Chili!Soft ASP for Solaris" in this chapter.
- Back up your existing installation prior to beginning the upgrade.
- The new installation will completely overwrite your existing installation. If there are any files you want to preserve in your current Sun Chili!Soft ASP installation directory, copy them to another location. Otherwise, they will be lost.

- Do not upgrade a mission-critical production server. Performing the upgrade and taking the additional configuration steps required might keep your server offline for an unacceptably long period of time. If possible, you should mirror the production server to a nonproduction server, perform the upgrade, and then test and debug the upgraded server before deploying it in your production environment.

When the setup program has completed the installation, do the following:

- Verify that any system DSNs defined for your previous installation are functioning correctly. For more information, see "Testing a DSN" in "Chapter 3: Managing Sun Chili!Soft ASP." If they are not functioning, you can create or edit the system DSNs as needed, as described in "Configuring Data Source Names (DSNs)" in "Chapter 3: Managing Sun Chili!Soft ASP." You can find the configuration information for the system DSNs that were defined for your previous installation in the following file:

  [C-ASP_INSTALL_DIR]/casp/INSTALL/settings.import

  where [C-ASP_INSTALL_DIR] is the path name of the Sun Chili!Soft ASP 3.6.2 installation directory.

Installing Sun Chili!Soft ASP for HP-UX

This section describes the requirements and procedures for installing Sun Chili!Soft ASP 3.6.2 for HP-UX. For the most up-to-date information about this product, see the README file included with your software. You can access the README file from the Sun Chili!Soft ASP Administration Console, as described in "Viewing the Product README File" in "Chapter 3: Managing Sun Chili!Soft ASP."

In this section:

- Installation Requirements: Sun Chili!Soft ASP for HP-UX
- Running the Setup Program: Sun Chili!Soft ASP for HP-UX
- Important Notes about HP-UX Installations
- Upgrading Sun Chili!Soft ASP for HP-UX

Installation Requirements: Sun Chili!Soft ASP for HP-UX

This section lists the hardware and software requirements for installing and running Sun Chili!Soft ASP 3.6.2 for HP-UX. It also lists additional software (such as databases and tools) that Sun Chili!Soft ASP supports.

Hardware and Software Requirements

Hardware

Your hardware configuration must meet the following minimum requirements:

- 256 MB RAM

- Pentium-class processor (x86)
- 140 MB free hard disk space

Operating System

Sun Chili!Soft ASP for HP-UX runs on the following operating system:

- HP-UX 11.0

Certain runtime library patches for the HP-UX operating system must be installed. For best performance, it is strongly recommended that you install the latest Quality Pack bundles (formerly known as General Release bundles). Quality Pack bundles are tested sets of HP-UX core patches that keep your operating system up to date.

You can obtain the Quality Pack bundles from HP at:

http://www.software.hp.com/SUPPORT_PLUS/

After you have installed the latest Quality Pack bundles, make sure that the patches listed below are installed on your machine. If any of the following required patches are not installed, you must manually download and install them.

The individual patches can be obtained from HP at:

http://us-support.external.hp.com/common/bin/doc.pl/

The required patches for HP-UX 11.0 are as follows:

Note

> This list does not include dependency patches. On the Web page from which you download the patch, be sure to click the "dependency" link and install the dependency patches.

- PHCO_23963 (s700_800 11.00 libc cumulative header file patch)
- PHCO_25707 (s700_800 11.00 libc cumulative patch)
- PHCO_24963 (s700_800 11.00 pthread(3t) cumulative man page patch)
- PHCO_26000 (s700_800 11.00 libpthreads cumulative patch)
- PHSS_24627 (s700_800 11.X HP aC++ -AA runtime libraries (aCC A.03.33))
- PHSS_22478 (s700_800 ld(1) and linker tools cumulative patch)
- PHKL_26059 (s700_800 11.00 syscall, signal, umask cumulative patch)
- PHKL_18543 - (PM/VM/UFS/async/scsi/io/DMAPI/JFS/perf patch)

Kernel Configuration

On the HP-UX 11.00 machine the default kernel configurable parameter is set too low and therefore is not sufficient to reliably run Sun Chili!Soft ASP. To increase the performance of this product, you should modify the maxdsiz configurable parameter. By default, this value is set to 64 MB. Depending upon your stress requirements, this value should be increased to at least 256

MB. Refer to your operating system documentation for information about how to increase this value.

Web Server

You must be running one of the following Web servers:

- HP Apache-based Web Server 1.3.19.23 DSO
- Apache 1.3.19 DSO
- iPlanet Web Server, Enterprise Edition 6.0 SP1

Note

If you want to install Sun Chili!Soft ASP 3.6.2 to the HP Apache-based Web server, you may first need to install that Web server. For more information and the download, go to:

http://www.software.hp.com/cgi-bin/swdepot_parser.cgi/cgi/displayProductInfo.pl?productNumber=B9415AAPA1319

Sun Chili!Soft ASP 3.6.2 may install to other versions of the supported Web servers listed above (Apache 1.3.12, for instance). However, versions not listed have not been certified to run with Sun Chili!Soft ASP 3.6.2, and their use is not supported by Sun Chili!Soft Customer Support.

Supported Software

This section lists additional software that is supported and/or installed by Sun Chili!Soft ASP.

Database

Sun Chili!Soft ASP for HP-UX includes ODBC drivers for the following databases:

DataDirect Connect ODBC 4.0 (Wire Protocol drivers)

- DB2 Universal Database (UDB) v7.1
- dBASE 5
- Informix Dynamic Server 9.x
- Informix Dynamic Server 2000 (9.20)
- Microsoft SQL Server 7.0 SP1
- Microsoft SQL Server 2000 SP1
- Oracle 8i (8.1.6)
- Oracle 8i (8.1.7)
- Oracle 9i
- Sybase Adaptive Server Enterprise 11.9.2
- Sybase Adaptive Server Enterprise 12.5
- Text Files

DataDirect SequeLink 4.5.1a

- Microsoft SQL Server 6.5 (via SequeLink)
- Microsoft Access 2000, 97, and 95 (via SequeLink)

ODBC drivers are installed automatically by the Sun Chili!Soft ASP setup program. Following installation, you must configure the drivers for the data source being used. For more information, see "Configuring a Database" in "Chapter 3: Managing Sun Chili!Soft ASP."

Note

To provide remote database connectivity with Microsoft Access and Microsoft SQL Server 6.5 databases, Sun Chili!Soft ASP ships with the client portion of the DataDirect SequeLink 4.51a software. On the Windows machine that contains your database, you also must download and install the server portion of this software from Sun Chili!Soft. For more information, see "Configuring SequeLink" in "Chapter 3: Managing Sun Chili!Soft ASP."

Sun Chili!Soft ASP 3.6.2 may work with other versions of the databases listed above. However, versions not listed have not been tested to run with Sun Chili!Soft ASP 3.6.2, and their use is not supported by Sun Chili!Soft Customer Support.

Java Runtime Environment

Sun Chili!Soft ASP 3.6.2 includes the Java Runtime Environment (JRE) 1.3.1 for Sun Chili!Beans. While JREs 1.2.x and 1.3.x are supported, and the Java 2 Runtime Environment can also be used, the use of JRE 1.3.1 is strongly recommended. Chili!Beans enables you to use Java class files as components called from VBScript or JScript. To use Chili!Beans, a Java runtime environment must be installed on the machine, and Chili!Beans must be enabled from the Administration Console. For more information, see "Chili!Beans Component Reference" in "Chapter 5: Developer's Reference."

Before installing Chili!Soft ASP 3.6.2, you must have the Java Runtime Environment (JRE) patches for HP-UX 11.00 installed, which you can obtain at:

http://www.unix.hp.com/java/infolibrary/patches.html

Microsoft FrontPage 2000 Server Extensions

Sun Chili!Soft ASP enables you to run ASP pages generated by Microsoft FrontPage. Microsoft FrontPage Server Extensions provide services to the Web server that work in conjunction with Sun Chili!Soft ASP to provide FrontPage functionality to computers not running Microsoft Windows NT or Windows 2000 and Internet Information Services (IIS).

FrontPage Server Extensions are no longer installed with Sun Chili!Soft ASP; you must obtain them from Microsoft. See "Enabling FrontPage Publishing" in "Chapter 3: Managing Sun Chili!Soft ASP."

Note

Specific questions about the installation, configuration, and use of FrontPage and FrontPage Server Extensions should be directed to Microsoft or its representatives.

Running the Setup Program: Sun Chili!Soft ASP for HP-UX

The Sun Chili!Soft ASP setup program takes you through the steps required to install Sun Chili!Soft ASP 3.6.2 on HP-UX. The setup program installs the following components on the target server:

- Sun Chili!Soft ASP Server
- Sun Chili!Soft ASP Administration Console
- Sun Chili!Beans support (the use of Chili!Beans is optional). Sun Chili!Soft ASP 3.6.2 includes Java Runtime Environment (JRE) 1.3.1. The use of this specific JRE version is not required, but is strongly recommended.
- Sun SpicePack components
- Apache Web Server 1.3.19 DSO (preconfigured or "bundled" with Sun Chili!Soft ASP 3.6.2).

During installation you can either specify the configuration options, or accept the default configuration settings. The configuration settings you specify during installation can have serious consequences for your server environment. Before you begin, make sure you meet the following requirements:

- You are logged in as root.
- You have 60 MB of hard disk space available to extract the installation files.
- Your hardware and software meet the requirements listed in "Installation Requirements: Sun Chili!Soft ASP for HP-UX" in this chapter.
- You know your product serial number. If you do not know your product serial number and you go ahead with the installation, you will receive an evaluation license.
- You know which language you want Sun Chili!Soft ASP to support (English - US, Japanese Shift-JIS, and so on) . If desired, you can change the language after installation, as described in "Configuring International Support" in "Chapter 3: Managing Sun Chili!Soft ASP."
- To use Chili!Beans, a Java runtime environment (JRE) must be installed on the machine, and Chili!Beans must be enabled from the Administration Console. During installation, Sun Chili!Soft ASP will install JRE 1.3.1 unless you choose to install another supported version (1.2.x or 1.3.x). Note that while you have the option to install a different JRE, and the Java 2 Runtime Environment can also be used, the use of JRE 1.3.1 is strongly recommended. Chili!Beans enables you to access Java objects and classes from an ASP script. For more information, see "Chili!Beans Component Reference" in "Chapter 5: Developer's Reference."
- If you are upgrading from a previous version of Sun Chili!Soft ASP, you must install Sun Chili!Soft ASP 3.6.2 in the same directory as your previous installation. Also, be sure to review the information in "Upgrading Sun Chili!Soft ASP for HP-UX" in this chapter.
- If you choose to customize settings for the Sun Chili!Soft ASP Administration Console, you must specify a username and password for accessing the console. Be sure to specify a

password that you can remember, or else plan to store the password in a secure location. If you forget your password, you can run the admtool utility and set a new one, as described in "Configuring Usernames and Passwords" in "Chapter 3: Managing Sun Chili!Soft ASP." If you do not choose the customize settings option, the username is set as "admin" and the password as "root." To protect the security of your server, you should run the admtool and change these as soon as possible following installation.

To run the setup program and install Sun Chili!Soft ASP, use the following procedure.

To run the setup program

1. If you are installing Sun Chili!Soft ASP 3.6.2 from the CD-ROM, go to step 2. If you are downloading Sun Chili!Soft ASP 3.6.2 from the Web, download the casp-3.6.2-HPUX.tar file to a temporary directory, and then extract the installation files by using the following command:

   ```
   #> tar -xvf casp-3.6.2-HPUX.tar
   ```

   This step creates the following files in the temporary directory, which you can delete after completing the installation:

   A QuickStart Guide with installation instructions
   A README file containing important product information
   The install.sh installation script
   A package/EULA file containing the End-User License Agreement
   A package directory containing files required to install Sun Chili!Soft ASP

2. Run the install.sh script by using the following command. For downloaded versions, this script is located in the temporary directory you created in the previous step. For CD-ROM versions, this script is located on your installation CD-ROM.

   ```
   #> ./install.sh
   ```

3. Review the End-User License Agreement (EULA) that displays. Enter **yes** if you agree with its conditions. You must type the entire word, "yes."

   – or –

   Enter **no** if you do not agree. If you enter **no**, the Sun Chili!Soft ASP setup program quits the installation.

4. On the next screen, press **Enter** to accept the default Sun Chili!Soft ASP installation directory (/opt/casp).

   – or –

   Enter the absolute path name of a different installation directory. Make note of this location, so you can easily find the Sun Chili!Soft ASP files at a later time.

   Note: To upgrade from the previous version, you must specify the installation directory of your existing installation.

5. If the setup program detects a previous installation of Sun Chili!Soft ASP in the specified directory, you will be prompted to uninstall the previous version (if a previous installation is not detected, go to step 6):

   Enter **n** (no) to change the installation directory or cancel the installation.

   – or –

   Enter **y** (yes) to uninstall the previous installation. You will then be prompted to proceed. At this confirmation prompt, enter **n** to cancel the installation, or **y** to uninstall the previous version. If you enter **y**, the previous version will be uninstalled by the installer. Once that process is finished, press **Enter** to continue.

   Note: At the end of the Sun Chili!Soft ASP installation process you will be prompted to import your settings.

6. On the next screen, enter **y** (yes) if you know the product serial number.

   – or –

   If you do not know the serial number, enter **n**. An evaluation license will be installed (go to step 8).

   Note: iPlanet 6.0 users are already licensed to use this product and do not need to enter a serial number. If you are using iPlanet 6.0, enter **n** and you will receive a full, unlimited license (go to step 8).

7. When prompted, enter the product serial number.

8. Sun Chili!Soft ASP 3.6.2 includes a ready-to-run Apache 1.3.19 Web server configured with support for Microsoft FrontPage 2002 Server Extensions (the extensions themselves are not installed) and EAPI (Extended API). If you have not yet configured a Web server, you have the option to install this preconfigured Apache Web server.

   Enter **y** (yes) to install the preconfigured ("bundled") Apache 1.3.19 Web server, and proceed to the next step.

   – or –

   Enter **n** (no) if you do not want to install the preconfigured Apache Web server, and proceed to step 10.

   Note: Installing the preconfigured Apache Web server will not affect any other installed Web server. However, make sure you do not install the preconfigured Apache Web server to the same port as that of an existing Web server.

9. The Apache Web server has many settings that you can select and configure. Specify the desired configuration option for the preconfigured Apache 1.3.19 Web server:

   Enter **1** (**Default configuration**) to direct Sun Chili!Soft ASP to automatically configure all settings for you.

   – or –

   Enter **2** (**Specify only the Web server listen port**) to specify only the port number that the Apache Web server will listen on, and then specify the port.

– or –

Enter **3** (**Customize the configuration**) to specify the configuration of many different settings, and then respond to the prompts.

Note: All Apache configuration settings can be changed manually after installation by editing the Apache configuration file.

10. On the next screen, press **Enter** to accept the default language (English – US).

    – or –

    Enter the number of a language shown in the list.

11. On the next screen, enable or disable Chili!Beans support as desired:

    Select **Use the bundled 1.3.1 JRE** to enable Chili!Beans support and install JRE 1.3.1 (this is the recommended JRE), and then respond to the prompts.

    – or –

    Select **Specify the path to an existing JRE** to specify the path to an existing JRE (JRE versions 1.2.x and 1.3.x are supported), or type **none** to return to the previous screen, and then respond to the prompts.

    – or –

    Select **Disable Java support** to disable Chili!Beans support, and then respond to the prompts.

    – or –

    Select **Keep your current settings**.

12. The next screen provides options for choosing the Web server to configure with Sun Chili!Soft ASP. If you want the setup program to search your system and generate a list of installed Web servers from which to choose, enter the number of the desired search option (**1**, **2**, or **3**). If you select one of these search options, skip to step 15.

    – or –

    If you want to skip this search and provide information about the Web server, enter **4** (**Don't search**). The installer will not search for the Web server, and instead takes you to step 13.

    Note: If you chose to install the preconfigured ("bundled") Apache 1.3.19 Web server in step 8, go to step 15.

13. On the next screen, enter the number of the type of Web server to configure: **1** for Apache, **2** for iPlanet, or **3** for Zeus, and go to step 14.

    Note: Zeus Web servers are not supported by Sun Chili!Soft ASP for HP-UX. You may be able to install Sun Chili!Soft ASP to a Zeus Web server, but that installation will not be supported by Sun Chili!Soft Customer Support.

    – or –

To cancel the user-specified Web server configuration, enter **4**. If you choose this option, the setup program returns you to the previous screen. From this screen you can choose another option, as described in step 15.

14. At the prompts, enter the path of your Web server. If you do not know the correct path, type **none** to cancel (this takes you to step 15). If you selected Apache in step 13, you will be prompted for the Web server binary. After all information has been added correctly, the installer takes you to step 15. There you will be able to select the Web server that was just added.

15. On the next screen, enter the number of the Web server you want the setup program to configure to run with Sun Chili!Soft ASP, and go to step 16 (if you selected option **1**, **2**, or **3** in step 12).

    – or –

    To provide information about the Web server to configure, enter the number for the option **Specify the Web server**. If you choose this option, follow the instructions in step 13.

    – or –

    To perform another search for installed Web servers, enter the number for the option **Attempt to auto-detect more Web servers**. If you choose this option, follow the instructions in step 12.

    – or –

    If you do not want to configure a Web server during installation, enter the number for the option **Do not configure a Web server**, and then go to step 18. If you choose this option, the first time you open the Sun Chili!Soft ASP Administration Console you will be prompted to configure the ASP Server and a Web server.

16. On the next screen, verify the information that the setup program displays about your Web server. If the information is incorrect, the Sun Chili!Soft ASP installation could fail. If the installation does fail, you can run the installation script again. If you do this, it is recommended that you first run the uninstall script, as described in "Uninstalling Sun Chili!Soft ASP for UNIX and Linux" in this chapter. If the information on the screen is correct, enter **y** (yes), and go to step 17.

    – or –

    If the information is incorrect, enter **n** (no). The setup program returns you to the previous screen (step 15).

17. On the next screen, choose a configuration option for the ASP Server:

    Enter **1** to choose a default configuration, and go to step 18.

    – or –

    Enter **2** to choose a custom configuration, and enter the following information at the prompts:

    Enable samples on this Web server? Enter **y** (yes) if you want the setup program to install and configure the Sun Chili!Soft ASP samples on your Web server, or enter **n** (no).

Enable documentation on this Web server? Enter **y** (yes) if you want the setup program to enable the Sun Chili!Soft ASP documentation on your Web server, or enter **n** (no). If you enter **n**, the documentation will be available from the Administration Console, but not from the Sun Chili!Soft ASP Start Page.

Start the ASP Server on system boot? Enter **y** (yes) if you want the setup program to start Sun Chili!Soft ASP automatically each time you start the computer, or enter **n** (no).

Would you like this [Web server] restarted? Enter **y** (yes) if you want the setup program to automatically restart the Web server after completing Sun Chili!Soft ASP installation, or enter **n** (no).

– or –

Enter **3** to choose another Web server to configure for Sun Chili!Soft ASP, and follow the instructions in step 12.

18. On the next screen, select a configuration option for the Administration Console. When you choose **Default configuration**, the setup program configures the console using default settings. With this option, the username is specified as "admin" and the password as "root." To protect the security of your server, it is strongly recommended that you change these settings as soon as possible. For information about doing this, see "Configuring Usernames and Passwords" in "Chapter 3: Managing Sun Chili!Soft ASP."

    If you choose **Custom configuration**, enter the following information at the prompts:

    Administration Console port number: Enter the number of the port on which the Web server should listen for requests for the Administration Console, or press **Enter** to accept the default.

    Automatically start the Administration Console on system startup: Enter **y** (yes) if you want the Sun Chili!Soft ASP Administration Console to start automatically each time you start the computer, or **n** (no).

    Type the username…: Enter the administrator username.

    New password: Enter the administrator password, which is required to access the Administration Console. Make note of this password, because you cannot access the Administration Console without providing it.

    Confirm password: Reenter the password.

19. The next screen provides summary information about your Administration Console configuration. Make note of this information or else print this screen for future reference. When finished, press **Enter** to complete the installation.

20. If you are upgrading from the previous version for HP-UX and the setup program was able to export the settings from your previous installation, you will be prompted to import these settings into your new installation. Enter **y** (yes) to import your settings.

    – or –

    Enter **n** (no) if you do not want to import your settings.

The setup program then completes the installation and writes a summary file to:

[C-ASP_INSTALL_DIR]/logs/install_summary

where [C-ASP_INSTALL_DIR] is the directory in which you installed Sun Chili!Soft ASP (/opt/casp by default). Be sure to print the information contained in the install_summary file for future reference.

Before running your installation of Sun Chili!Soft ASP 3.6.2, see "Important Notes about HP-UX Installations" in this chapter. If you are upgrading from a previous version of Sun Chili!Soft ASP, see "Upgrading Sun Chili!Soft ASP for HP-UX" in this chapter.

For information about changes the setup program makes to your Web server configuration files, see "Changes to Web Server Configuration Files" in this chapter. For information about how to change some of the options that were configured during installation, see "Changing Installation Options after Installation" in this chapter.

See also:

Uninstalling Sun Chili!Soft ASP for UNIX and Linux in this chapter

Important Notes about HP-UX Installations

Before running your installation of Sun Chili!Soft ASP 3.6.2 for HP-UX, review the following information:

- If you chose the default configuration for the Administration Console, the username is configured as "admin" and the password as "root." To protect the security of your server, it is strongly recommended that you change the username and password as soon as possible. For more information, see "Configuring Usernames and Passwords" in "Chapter 3: Managing Sun Chili!Soft ASP."
- You use the Administration Console to start and stop the Sun Chili!Soft ASP Server and to change configuration settings. You can access the Administration Console by typing the following URL in your Web browser address bar:

  http://[HOSTNAME]:[PORT]

  where [HOSTNAME] is the valid DNS name or IP address of the Apache Web server running the Sun Chili!Soft ASP Administration Console, and [PORT] is the port on which the Administration Console is configured to run (5100 by default). The URL for accessing the Administration Console is provided in the following file:

  [C-ASP_INSTALL_DIR]/logs/install_summary

  where [C-ASP_INSTALL_DIR] is the directory in which you installed Sun Chili!Soft ASP (/opt/casp by default).
- If you chose not to configure a Web server to run with Sun Chili!Soft ASP during installation, the first time you open the Administration Console you will be prompted to install a Web server.
- The Sun Chili!Soft ASP setup program makes certain changes to the configuration files for the associated Web server. For more information, see "Changes to Web Server Configuration Files" in this chapter.

- Following installation, you can change some of the options that were configured during installation, such as the Web server with which Sun Chili!Soft ASP is configured to run, and the status of Java support (enabled or disabled). For more information, see "Changing Installation Options after Installation" in this chapter.
- To use Chili!Beans, a Java runtime environment (JRE) must be installed on the machine, and Chili!Beans must be enabled from the Administration Console (JRE 1.3.1 is included with Sun Chili!Soft ASP 3.6.2 and is the recommended version). If you did not install a JRE during the installation of Sun Chili!Soft ASP, you can do so by following the instructions in "Enabling Java Support" in this chapter.
- When finished installing Sun Chili!Soft ASP 3.6.2, use the diagnostic applications to verify that your installation is functioning correctly. You can access the diagnostics from the following URL:

  http://[HOSTNAME]/caspsamp/diagnostics.htm

  where [HOSTNAME] is the hostname of the Web server configured to run with Sun Chili!Soft ASP.
- Important summary information about the installation is located in the following file:

  [C-ASP_INSTALL_DIR]/logs/install_summary

  where [C-ASP_INSTALL_DIR] is the directory in which you installed Sun Chili!Soft ASP (/opt/casp by default).

  Be sure to print this information for future reference.
- Certain Sun Chili!Soft ASP 3.6.2 settings have important implications for the security of the Sun Chili!Soft ASP Server. See "Securing the Server" in "Chapter 3: Managing Sun Chili!Soft ASP" to ensure that the settings are appropriately configured for your specific environment.

See also:

Uninstalling Sun Chili!Soft ASP for UNIX and Linux in this chapter

Upgrading Sun Chili!Soft ASP for HP-UX

Sun Chili!Soft ASP 3.6.2 automatically upgrades many of your settings from an existing 3.6 installation of Sun Chili!Soft ASP for HP-UX.

To perform an upgrade from Sun Chili!Soft ASP 3.6 to Sun Chili!Soft ASP 3.6.2, follow the instructions in "Running the Setup Program: Sun Chili!Soft ASP for HP-UX" in this chapter. Also, review the following information and take any necessary steps:

- The Sun Chili!Soft ASP 3.6.2 setup program preserves the settings from your Sun Chili!Soft ASP 3.6 installation, and then imports these settings into your new Sun Chili!Soft ASP 3.6.2 installation. The settings imported from your existing installation will override the settings that you configure during installation, with the exception of Chili!Beans support. During installation, you must configure Chili!Beans support and

select the option for the bundled JRE 1.3.1 (or specify a path to a different JRE). While JREs 1.2.x and 1.3.x are supported, and the Java 2 Runtime Environment can also be used, the use of JRE 1.3.1 is strongly recommended. Java support must be configured, even if it was enabled in the installation you are upgrading.

Note

SpicePack settings and Sybase DSNs are not migrated when upgrading to Sun Chili!Soft ASP 3.6.2.

- If necessary, upgrade your Web server to meet the requirements listed in "Installation Requirements: Sun Chili!Soft ASP for HP-UX" in this chapter.
- Back up your existing installation prior to beginning the upgrade.
- The new installation will completely overwrite your existing installation. If there are any files you want to preserve in your current Sun Chili!Soft ASP installation directory, copy them to another location. Otherwise, they will be lost.
- Do not upgrade a mission-critical production server. Performing the upgrade and taking the additional configuration steps required might keep your server offline for an unacceptably long period of time. If possible, you should mirror the production server to a nonproduction server, perform the upgrade, and then test and debug the upgraded server before deploying it in your production environment.

When the setup program has completed the installation, do the following:

- Verify that any system DSNs defined for your previous installation are functioning correctly. For more information, see "Testing a DSN" in "Chapter 3: Managing Sun Chili!Soft ASP." If they are not functioning, you can create or edit the system DSNs as needed, as described in "Configuring Data Source Names (DSNs)" in "Chapter 3: Managing Sun Chili!Soft ASP." You can find the configuration information for the system DSNs that were defined for your previous installation in the following file:

  [C-ASP_INSTALL_DIR]/casp/INSTALL/settings.import

  where [C-ASP_INSTALL_DIR] is the path name of the Sun Chili!Soft ASP 3.6.2 installation directory.

Installing Sun Chili!Soft ASP for Linux

This section describes the requirements and procedures for installing Sun Chili!Soft ASP 3.6.2 for Linux. For the most up-to-date information about this product, see the README file included with your software. You can access the README file from the Sun Chili!Soft ASP Administration Console, as described in "Viewing the Product README File" in "Chapter 3: Managing Sun Chili!Soft ASP."

In this section:

- Installation Requirements: Sun Chili!Soft ASP for Linux

- Running the Setup Program: Sun Chili!Soft ASP for Linux
- Important Notes about Linux Installations
- Upgrading Sun Chili!Soft ASP for Linux

Installation Requirements: Sun Chili!Soft ASP for Linux

This section lists the hardware and software requirements for installing and running Sun Chili!Soft ASP 3.6.2 for Linux. It also lists the additional software (such as databases and tools) that Sun Chili!Soft ASP supports.

Hardware and Software Requirements

Hardware

Your hardware configuration must meet the following minimum requirements:

- 256 MB RAM
- Pentium-class processor (x86)
- 140 MB free hard disk space

Operating System

Sun Chili!Soft ASP for Linux runs on the following operating systems:

- Red Hat Linux 7.2 (2.4 kernel, GLIBC 2.2.4))
- SuSE 7.3 Professional (2.4 kernel, GLIBC 2.2.4)
- Mandrake Linux 8.1 (2.4 kernel, GLIBC 2.2.4)
- Debian 2.2r5 (2.2 kernel, GLIBC 2.1.3)

Web Server

You must be running one of the following Web servers:

- Apache 1.3.19 DSO
- Apache 1.3.22 DSO
- iPlanet Web Server, Enterprise Edition 6.0 SP1
- Zeus Web Server 4.0

Note

Sun Chili!Soft ASP 3.6.2 may install to other versions of the supported Web servers listed above (Apache 1.3.12, for example). However, versions not listed have not been certified to run with Sun Chili!Soft ASP 3.6.2, and their use is not supported by Sun Chili!Soft Customer Support.

Supported Software

This section lists additional software that is supported and/or installed by Sun Chili!Soft ASP.

Database

Sun Chili!Soft ASP for Linux includes ODBC drivers for the following databases:

DataDirect Connect ODBC 4.0 (Wire Protocol drivers)

- DB2 Universal Database (UDB) v7.1
- dBASE 5
- Informix Dynamic Server 2000 (9.20)
- Microsoft SQL Server 7.0 SP1
- Microsoft SQL Server 2000 SP1
- Oracle 8i (8.1.6)
- Oracle 8i (8.1.7)
- Oracle 9i
- Sybase Adaptive Server Enterprise 11.9.2
- Sybase Adaptive Server Enterprise 12.5
- Text Files

DataDirect SequeLink 4.5.1a

- Microsoft SQL Server 6.5 (via SequeLink)
- Microsoft Access 2000, 97, and 95 (via SequeLink)

Open Source

- MySQL 3.22.30 and 3.23.49
- PostgreSQL 6.5.2 and 7.1.3

ODBC drivers are installed automatically by the Sun Chili!Soft ASP setup program. Following installation, you must configure the drivers for the data source being used. For more information, see "Configuring a Database" in "Chapter 3: Managing Sun Chili!Soft ASP."

Note

To provide remote database connectivity with Microsoft Access and Microsoft SQL Server 6.5 databases, Sun Chili!Soft ASP ships with the client portion of the DataDirect SequeLink 4.51a software. On the Windows machine that contains your database, you also must download and install the server portion of this software from Sun Chili!Soft. For more information, see "Configuring SequeLink" in "Chapter 3: Managing Sun Chili!Soft ASP."

Sun Chili!Soft ASP 3.6.2 may work with other versions of the databases listed above. However, versions not listed have not been tested to run with Sun Chili!Soft ASP 3.6.2, and their use is not supported by Sun Chili!Soft Customer Support.

Java Runtime Environment

Sun Chili!Soft ASP 3.6.2 includes the Java Runtime Environment (JRE) 1.3.1 for Sun Chili!Beans. While JREs 1.2.x and 1.3.x are supported, and the Java 2 Runtime Environment can also be used, the use of JRE 1.3.1 is strongly recommended. Chili!Beans enables you to use Java class files as components called from VBScript or JScript. To use Chili!Beans, a Java runtime environment must be installed on the machine, and Chili!Beans must be enabled from the Administration Console. For more information, see "Chili!Beans Component Reference" in "Chapter 5: Developer's Reference."

Microsoft FrontPage Server Extensions

Sun Chili!Soft ASP enables you to run ASP pages generated by Microsoft FrontPage. Microsoft FrontPage Server Extensions provide services to the Web server that work in conjunction with Sun Chili!Soft ASP to provide FrontPage functionality to computers not running Microsoft Windows NT or Windows 2000 and Internet Information Services (IIS).

FrontPage Server Extensions are no longer installed with Sun Chili!Soft ASP; you must obtain them from Microsoft. See "Enabling FrontPage Publishing" in "Chapter 3: Managing Sun Chili!Soft ASP."

Note

> Specific questions about the installation, configuration, and use of FrontPage and FrontPage Server Extensions should be directed to Microsoft or its representatives.

See also:

Installing Sun Chili!Soft ASP for Linux in this chapter

Running the Setup Program: Sun Chili!Soft ASP for Linux

The Sun Chili!Soft ASP setup program takes you through the steps required to install Sun Chili!Soft ASP 3.6.2 on Linux. The setup program installs the following components on the target server:

- Sun Chili!Soft ASP Server
- Sun Chili!Soft ASP Administration Console
- Sun Chili!Beans support (the use of Chili!Beans is optional). Sun Chili!Soft ASP 3.6.2 includes Java Runtime Environment (JRE) 1.3.1. The use of this specific JRE version is not required, but is strongly recommended.
- Sun SpicePack components
- Apache Web Server 1.3.19 DSO (preconfigured or "bundled" with Sun Chili!Soft ASP 3.6.2).

During installation you can either specify the configuration options, or accept the default configuration settings. The configuration settings you specify during installation can have serious

consequences for your server environment. Before you begin, make sure you meet the following requirements:

- You are logged in as root.
- You have 60 MB of hard disk space available to extract the installation files.
- Your hardware and software meet the requirements listed in "Installation Requirements: Sun Chili!Soft ASP for Linux" in this chapter.
- You know your product serial number. If you do not know your product serial number and you go ahead with the installation, you will receive an evaluation license.
- You know which language you want Sun Chili!Soft ASP to support (English - US, Japanese Shift-JIS, and so on) . If desired, you can change the language after installation, as described in "Configuring International Support" in "Chapter 3: Managing Sun Chili!Soft ASP."
- To use Chili!Beans, a Java runtime environment (JRE) must be installed on the machine, and Chili!Beans must be enabled from the Administration Console. During installation, Sun Chili!Soft ASP will install JRE 1.3.1 unless you choose to install another supported version (1.2.x or 1.3.x). Note that while you have the option to install a different JRE, and the Java 2 Runtime Environment can also be used, the use of JRE 1.3.1 is strongly recommended. Chili!Beans enables you to access Java objects and classes from an ASP script. For more information, see "Chili!Beans Component Reference" in "Chapter 5: Developer's Reference."
- If you are upgrading from a previous version of Sun Chili!Soft ASP, you must install Sun Chili!Soft ASP 3.6.2 in the same directory as your previous installation. Also, be sure to review the information in "Upgrading Sun Chili!Soft ASP for Linux" in this chapter.
- If you choose to customize settings for the Sun Chili!Soft ASP Administration Console, you must specify a username and password for accessing the console. Be sure to specify a password that you can remember, or else plan to store the password in a secure location. If you forget your password, you can run the admtool utility and set a new one, as described in "Configuring Usernames and Passwords" in "Chapter 3: Managing Sun Chili!Soft ASP." If you do not choose the customize settings option, the username is set as "admin" and the password as "root." To protect the security of your server, you should run the admtool and change these as soon as possible following installation.
- If you are installing Sun Chili!Soft ASP to a Zeus Web server, you must take additional steps prior to installation. See "Installing Sun Chili!Soft ASP to a Zeus Web Server" immediately below. If you are not installing to a Zeus Web server, proceed with the installation as described in "To run the setup program" in this section.

Installing Sun Chili!Soft ASP to a Zeus Web Server

Before you can install Sun Chili!Soft ASP to a Zeus Web server, you must first enable NSAPI (Netscape’s Web server API) for a Zeus Virtual Server, and then copy the NSAPI configuration files to the proper location. For detailed information beyond what is provided here, see your Zeus product documentation.

Note

When configuring the Zeus Web server to run with Sun Chili!Soft ASP 3.6.2, do not specify the Web server hostname as the fully qualified domain name (or "server address"). If you do, the Test DSN feature of the Administration Console will not work. Instead, use the hostname only. For example, do not use: hostname.domain.com. Instead, use: hostname.

To enable NSAPI for a Zeus Virtual Server

1. Go to the **Configuration Summary** page for the Zeus Virtual Server.
2. Go to the **Enabling NSAPI Functionality** page by clicking the **NSAPI** link.
3. Click the **Enabled** button, write down the value of `[path_prefix]` for future reference, and then click the **Apply changes** button.
4. Commit the changes by clicking the yellow sun graphic in the top right section of the page, and then click the **Commit** button.

   NSAPI has been enabled for a Zeus Virtual Server. You must now copy the NSAPI configuration files, as described in the following procedure.

To copy the NSAPI configuration files

Zeus provides copies of the NSAPI configuration files, which are typically located in the following directory:

```
[zeushome]/webadmin-4.0r1/docroot/nsapi-skel/
```

where [zeushome] is the root of your Zeus installation (that is, /usr/local/zeus).

Once the files have been located, use the following commands:

```
#> mkdir –p [path_prefix]/https-[virtual_server_name]/config
#> cd [path_prefix]/https-[virtual_server_name]/config
#> cp –r [zeushome]/webadmin-4.0r1/docroot/nsapi-skel/* .
```

where `[path_prefix]` is the value you wrote down in step 3 of the previous procedure (the default will be `[zeushome]/ns-config`), and `[virtual_server_name]` is the name of the Zeus Virtual Server. Zeus is now configured correctly for the installation of Sun Chili!Soft ASP.

To run the setup program and install Sun Chili!Soft ASP, use the following procedure.

To run the setup program

1. If you are installing Sun Chili!Soft ASP 3.6.2 from the CD-ROM, go to step 2. If you are downloading Sun Chili!Soft ASP 3.6.2 from the Web, download the casp-3.6.2-linux.tar file to a temporary directory, and then extract the installation files by using the following command:

   ```
   #> tar -xvf casp-3.6.2-linux.tar
   ```

This step creates the following files in the temporary directory, which you can delete after completing the installation:

A QuickStart Guide with installation instructions
A README file containing important product information
The install.sh installation script
A package/EULA file containing the End-User License Agreement
A package directory containing files required to install Sun Chili!Soft ASP

2. Run the install.sh script by using the following command. For downloaded versions, this script is located in the temporary directory you created in the previous step. For CD-ROM versions, this script is located on your installation CD-ROM.

```
#> ./install.sh
```

3. Review the End-User License Agreement (EULA) that displays. Enter **yes** if you agree with its conditions. You must type the entire word, "yes."

   – or –

   Enter **no** if you do not agree. If you enter **no**, the Sun Chili!Soft ASP setup program quits the installation.

4. On the next screen, press **Enter** to accept the default Sun Chili!Soft ASP installation directory (/opt/casp).

   – or –

   Enter the absolute path name of a different installation directory. Make note of this location, so you can easily find the Sun Chili!Soft ASP files at a later time.

   Note: To upgrade from the previous version, you must specify the installation directory of your existing installation.

5. If the setup program detects a previous installation of Sun Chili!Soft ASP in the specified directory, you will be prompted to uninstall the previous version (if a previous installation is not detected, go to step 6):

   Enter **n** (no) to change the installation directory or cancel the installation.

   – or –

   Enter **y** (yes) to uninstall the previous installation. You will then be prompted to proceed. At this confirmation prompt, enter **n** to cancel the installation, or **y** to uninstall the previous version. If you enter **y**, the previous version will be uninstalled by the installer. Once that process is finished, press **Enter** to continue.

   Note: At the end of the Sun Chili!Soft ASP installation process you will be prompted to import your settings.

6. On the next screen, enter **y** (yes) if you know the product serial number.

   – or –

   If you do not know the serial number, enter **n**. An evaluation license will be installed (go to step 8).

Note: iPlanet 6.0 users are already licensed to use this product and do not need to enter a serial number. If you are using iPlanet 6.0, enter **n** and you will receive a full, unlimited license (go to step 8).

7. When prompted, enter the product serial number.
8. Sun Chili!Soft ASP 3.6.2 includes a ready-to-run Apache 1.3.19 Web server configured with support for Microsoft FrontPage 2002 Server Extensions (the extensions themselves are not installed) and EAPI (Extended API). If you have not yet configured a Web server, you have the option to install this preconfigured Apache Web server.

   Enter **y** (yes) to install the preconfigured ("bundled") Apache 1.3.19 Web server, and proceed to the next step.

   – or –

   Enter **n** (no) if you do not want to install the preconfigured Apache Web server, and proceed to step 10.

   Note: Installing the preconfigured Apache Web server will not affect any other installed Web server. However, make sure you do not install the preconfigured Apache Web server to the same port as that of an existing Web server.
9. The Apache Web server has many settings that you can select and configure. Specify the desired configuration option for the preconfigured Apache 1.3.19 Web server:

   Enter **1** (**Default configuration**) to direct Sun Chili!Soft ASP to automatically configure all settings for you.

   – or –

   Enter **2** (**Specify only the Web server listen port**) to specify only the port number that the Apache Web server will listen on, and then specify the port.

   – or –

   Enter **3** (**Customize the configuration**) to specify the configuration of many different settings, and then respond to the prompts.

   Note: All Apache configuration settings can be changed manually after installation by editing the Apache configuration file.
10. On the next screen, press **Enter** to accept the default language (English – US).

    – or –

    Enter the number of a language shown in the list.
11. On the next screen, enable or disable Chili!Beans support as desired:

    Select **Use the bundled 1.3.1 JRE** to enable Chili!Beans support and install JRE 1.3.1 (this is the recommended JRE), and then respond to the prompts.

    – or –

Select **Specify the path to an existing JRE** to specify the path to an existing JRE (JRE versions 1.2.x and 1.3.x are supported), or type **none** to return to the previous screen, and then respond to the prompts.

– or –

Select **Disable Java support** to disable Chili!Beans support, and then respond to the prompts.

– or –

Select **Keep your current settings**.

12. The next screen provides options for choosing the Web server to configure with Sun Chili!Soft ASP. If you want the setup program to search your system and generate a list of installed Web servers from which to choose, enter the number of the desired search option (**1**, **2**, or **3**). If you select one of these search options, skip to step 15.

    – or –

    If you want to skip this search and provide information about the Web server, enter **4** (**Don't search**). The installer will not search for the Web server, and instead takes you to step 13.

    Note: If you chose to install the preconfigured ("bundled") Apache 1.3.19 Web server in step 8, go to step 15.

13. On the next screen, enter the number of the type of Web server to configure: **1** for Apache, **2** for iPlanet, or **3** for Zeus, and go to step 14.

    – or –

    To cancel the user-specified Web server configuration, enter **4**. If you choose this option, the setup program returns you to the previous screen. From this screen you can choose another option, as described in step 15.

14. At the prompts, enter the path of your Web server. If you do not know the correct path, type **none** to cancel (this takes you to step 15). If you selected Apache in step 13, you will be prompted for the Web server binary. After all information has been added correctly, the installer takes you to step 15. There you will be able to select the Web server that was just added.

15. On the next screen, enter the number of the Web server you want the setup program to configure to run with Sun Chili!Soft ASP, and go to step 16 (if you selected option **1**, **2**, or **3** in step 12).

    – or –

    To provide information about the Web server to configure, enter the number for the option **Specify the Web server**. If you choose this option, follow the instructions in step 13.

    – or –

    To perform another search for installed Web servers, enter the number for the option **Attempt to auto-detect more Web servers**. If you choose this option, follow the instructions in step 12.

– or –

If you do not want to configure a Web server during installation, enter the number for the option **Do not configure a Web server**, and then go to step 18. If you choose this option, the first time you open the Sun Chili!Soft ASP Administration Console you will be prompted to configure the ASP Server and a Web server.

16. On the next screen, verify the information that the setup program displays about your Web server. If the information is incorrect, the Sun Chili!Soft ASP installation could fail. If the installation does fail, you can run the installation script again. If you do this, it is recommended that you first run the uninstall script, as described in "Uninstalling Sun Chili!Soft ASP for UNIX and Linux" in this chapter. If the information on the screen is correct, enter **y** (yes), and go to step 17.

    – or –

    If the information is incorrect, enter **n** (no). The setup program returns you to the previous screen (step 15).

17. On the next screen, choose a configuration option for the ASP Server:

    Enter **1** to choose a default configuration, and go to step 18.

    – or –

    Enter **2** to choose a custom configuration, and enter the following information at the prompts:

    Enable samples on this Web server? Enter **y** (yes) if you want the setup program to install and configure the Sun Chili!Soft ASP samples on your Web server, or enter **n** (no).

    Enable documentation on this Web server? Enter **y** (yes) if you want the setup program to enable the Sun Chili!Soft ASP documentation on your Web server, or enter **n** (no). If you enter **n**, the documentation will be available from the Administration Console, but not from the Sun Chili!Soft ASP Start Page.

    Start the ASP Server on system boot? Enter **y** (yes) if you want the setup program to start Sun Chili!Soft ASP automatically each time you start the computer, or enter **n** (no).

    Would you like this [Web server] restarted? Enter **y** (yes) if you want the setup program to automatically restart the Web server after completing Sun Chili!Soft ASP installation, or enter **n** (no).

    – or –

    Enter **3** to choose another Web server to configure for Sun Chili!Soft ASP, and follow the instructions in step 12.

18. On the next screen, select a configuration option for the Administration Console. When you choose **Default configuration**, the setup program configures the console using default settings. With this option, the username is specified as "admin" and the password as "root." To protect the security of your server, it is strongly recommended that you change these settings as soon as possible. For information about doing this, see "Configuring Usernames and Passwords" in "Chapter 3: Managing Sun Chili!Soft ASP."

If you choose **Custom configuration**, enter the following information at the prompts:

Administration Console port number: Enter the number of the port on which the Web server should listen for requests for the Administration Console, or press **Enter** to accept the default.

Automatically start the Administration Console on system startup: Enter **y** (yes) if you want the Sun Chili!Soft ASP Administration Console to start automatically each time you start the computer, or enter **n** (no).

Type the username…: Enter the administrator username.

New password: Enter the administrator password, which is required to access the Administration Console. Make note of this password, because you cannot access the Administration Console without providing it.

Confirm password: Reenter the password.

19. The next screen provides summary information about your Administration Console configuration. Make note of this information or else print this screen for future reference. When finished, press **Enter** to complete the installation.
20. If you are upgrading from the previous version for Linux and the setup program was able to export the settings from your previous installation, you will be prompted to import these settings into your new installation. Enter **y** (yes) to import your settings.

    – or –

    Enter **n** (no) if you do not want to import your settings.

The setup program then completes the installation and writes a summary file to:

[C-ASP_INSTALL_DIR]/logs/install_summary

where [C-ASP_INSTALL_DIR] is the directory in which you installed Sun Chili!Soft ASP (/opt/casp by default). Be sure to print the information contained in the install_summary file for future reference.

Note

With Sun Chili!Soft ASP for Linux on Zeus, you must access your Web server by using the "server address" specified in the Zeus Administration Console, rather than the address specified in the install_summary file.

Before running your installation of Sun Chili!Soft ASP 3.6.2, see "Important Notes about Linux Installations" in this chapter. If you are upgrading from a previous version of Sun Chili!Soft ASP for Linux, see "Upgrading Sun Chili!Soft ASP for Linux" in this chapter.

For information about changes the setup program makes to your Web server configuration files, see "Changes to Web Server Configuration Files" in this chapter. For information about how to change some of the options that were configured during installation, see "Changing Installation Options after Installation" in this chapter.

See also:

Uninstalling Sun Chili!Soft ASP for UNIX and Linux in this chapter

Important Notes about Linux Installations

Before running your installation of Sun Chili!Soft ASP 3.6.2 for Linux, review the following information:

- If you chose the default configuration for the Administration Console, the username is configured as "admin" and the password as "root." To protect the security of your server, it is strongly recommended that you change the username and password as soon as possible. For more information, see "Configuring Usernames and Passwords" in "Chapter 3: Managing Sun Chili!Soft ASP."
- You use the Administration Console to start and stop the Sun Chili!Soft ASP Server and to change configuration settings. You can access the Administration Console by typing the following URL in your Web browser address bar:

  http://[HOSTNAME]:[PORT]

  where [HOSTNAME] is the valid DNS name or IP address of the Apache Web server running the Sun Chili!Soft ASP Administration Console, and [PORT] is the port on which the Administration Console is configured to run (5100 by default). The URL for accessing the Administration Console is provided in the following file:

  [C-ASP_INSTALL_DIR]/logs/install_summary

  where [C-ASP_INSTALL_DIR] is the directory in which you installed Sun Chili!Soft ASP (/opt/casp by default).
- If you chose not to configure a Web server to run with Sun Chili!Soft ASP during installation, the first time you open the Administration Console you will be prompted to install a Web server.
- The Sun Chili!Soft ASP setup program makes certain changes to the configuration files for the associated Web server. For more information, see "Changes to Web Server Configuration Files" in this chapter.
- Following installation, you can change some of the options that were configured during installation, such as the Web server with which Sun Chili!Soft ASP is configured to run, and the status of Java support (enabled or disabled). For more information, see "Changing Installation Options after Installation" in this chapter.
- To use Chili!Beans, a Java runtime environment (JRE) must be installed on the machine, and Chili!Beans must be enabled from the Administration Console (JRE 1.3.1 is included with Sun Chili!Soft ASP 3.6.2 and is the recommended version). If you did not install a JRE during the installation of Sun Chili!Soft ASP, you can do so by following the instructions in "Enabling Java Support" in this chapter.
- When finished installing Sun Chili!Soft ASP 3.6.2, use the diagnostic applications to verify that your installation is functioning correctly. You can access the diagnostics from the following URL:

http://[HOSTNAME]/caspsamp/diagnostics.htm

where [HOSTNAME] is the hostname of the Web server configured to run with Sun Chili!Soft ASP.

- Important summary information about the installation is located in the following file:

  [C-ASP_INSTALL_DIR]/logs/install_summary

  where [C-ASP_INSTALL_DIR] is the directory in which you installed Sun Chili!Soft ASP (/opt/casp by default).

  Be sure to print this information for future reference.

Note

With Sun Chili!Soft ASP for Linux on Zeus, you must access your Web server by using the "server address" specified in the Zeus Administration Console, rather than the address specified in the install_summary file.

- Certain Sun Chili!Soft ASP 3.6.2 settings have important implications for the security of the Sun Chili!Soft ASP Server. See "Securing the Server" in "Chapter 3: Managing Sun Chili!Soft ASP" to ensure that the settings are appropriately configured for your specific environment.

See also:

Uninstalling Sun Chili!Soft ASP for UNIX and Linux in this chapter

Upgrading Sun Chili!Soft ASP for Linux

Sun Chili!Soft ASP 3.6.2 automatically upgrades many of your settings from an existing installation of Sun Chili!Soft ASP for Linux.

To perform an upgrade from Sun Chili!Soft ASP 3.6 to Sun Chili!Soft ASP 3.6.2, follow the instructions in "Running the Setup Program: Sun Chili!Soft ASP for Linux" in this chapter. Also, review the following information and take any necessary steps:

- The Sun Chili!Soft ASP 3.6.2 setup program preserves the settings from your Sun Chili!Soft ASP 3.6 installation, and then imports these settings into your new Sun Chili!Soft ASP 3.6.2 installation. The settings imported from your existing installation will override the settings that you configure during installation, with the exception of Chili!Beans support. During installation, you must configure Chili!Beans support and select the option for the bundled JRE 1.3.1 (or specify a path to a different JRE). While JREs 1.2.x and 1.3.x are supported, and the Java 2 Runtime Environment can also be used, the use of JRE 1.3.1 is strongly recommended. Java support must be configured, even if it was enabled in the installation you are upgrading.

Note

SpicePack settings and Sybase DSNs are not migrated when upgrading to Sun Chili!Soft ASP 3.6.2.

- If necessary, upgrade your Web server to meet the requirements listed in "Installation Requirements: Sun Chili!Soft ASP for Linux" in this chapter.
- Back up your existing installation prior to beginning the upgrade.
- The new installation will completely overwrite your existing installation. If there are any files you want to preserve in your current Sun Chili!Soft ASP installation directory, copy them to another location. Otherwise, they will be lost.
- Do not upgrade a mission-critical production server. Performing the upgrade and taking the additional configuration steps required might keep your server offline for an unacceptably long period of time. If possible, you should mirror the production server to a nonproduction server, perform the upgrade, and then test and debug the upgraded server before deploying it in your production environment.

When the setup program has completed the installation, do the following:

- Verify that any system DSNs defined for your previous installation are functioning correctly. For more information, see "Testing a DSN" in "Chapter 3: Managing Sun Chili!Soft ASP." If they are not functioning, you can create or edit the system DSNs as needed, as described in "Configuring Data Source Names (DSNs)" in "Chapter 3: Managing Sun Chili!Soft ASP." You can find the configuration information for the system DSNs that were defined for your previous installation in the following file:

  [C-ASP_INSTALL_DIR]/casp/INSTALL/settings.import

  where [C-ASP_INSTALL_DIR] is the path name of the Sun Chili!Soft ASP 3.6.2 installation directory.

Installing Sun Chili!Soft ASP for Windows

This section describes the requirements and procedures for installing Sun Chili!Soft ASP 3.6.2 for Microsoft Windows.

In this section:

- Installation Requirements: Sun Chili!Soft ASP for Windows
- Running the Setup Program: Sun Chili!Soft ASP for Windows
- Upgrading Sun Chili!Soft ASP for Windows

Installation Requirements: Sun Chili!Soft ASP for Windows

This section lists the hardware and software requirements for installing and running Sun Chili!Soft ASP 3.6.2 for Windows.

Hardware and Software Requirements

Hardware

Your hardware configuration must meet the following minimum requirements:

- 128 MB RAM (256 MB recommended)
- Pentium processor
- 100 MB free hard disk space

Operating Systems

Sun Chili!Soft ASP for Windows runs on the following operating systems:

- Microsoft Windows NT Server 4.0 SP6
- Microsoft Windows 2000 Server SP1

Web Servers

You must be running one of the following Web servers:

- Apache 1.3.22
- iPlanet Web Server, Enterprise Edition 6.0 SP1

Optional Requirements

Internet Explorer 5.0 or Internet Information Server

- Windows NT requirement: To use VBScript, JScript, the FileSystem object, and the BrowsCap component with Sun Chili!Soft ASP, Internet Explorer 5.0 or higher or any version of Internet Information Server (IIS) must be installed on your Web server.
- Windows 2000 requirement: To use VBScript and JScript with Sun Chili!Soft ASP, Internet Explorer (IE) 5.0 or higher or any version of Internet Information Server (IIS) must be installed on your Web server.

Microsoft Data Access Components (MDAC) 2.5

To install and use the Sun Chili!Soft ASP ADO samples, MDAC 2.5 (or higher) must be installed on your computer. If you opt to install the samples during the installation of Sun Chili!Soft ASP, and MDAC 2.5 is not yet installed, you have the option to install it.

Microsoft FrontPage Server Extensions

Sun Chili!Soft ASP enables you to run ASP pages generated by Microsoft FrontPage. Microsoft FrontPage Server Extensions provide services to the Web server that work in conjunction with Sun Chili!Soft ASP to provide FrontPage functionality to computers not running Microsoft Windows NT or Windows 2000 and Internet Information Services (IIS).

FrontPage Server Extensions are not installed with Sun Chili!Soft ASP; you must obtain them from Microsoft. Specific questions about the installation, configuration, and use of FrontPage and FrontPage Server Extensions should be directed to Microsoft or its representatives.

Note

ODBC drivers are not installed with Sun Chili!Soft ASP for Windows. Use the Windows ODBC Data Source Administrator (accessed from the Control Panel) to view installed ODBC drivers, and to create and manage DSNs.

See also:

Running the Setup Program: Sun Chili!Soft ASP for Windows in this chapter

Running the Setup Program: Sun Chili!Soft ASP for Windows

The Sun Chili!Soft ASP setup program takes you through the steps required to install Sun Chili!Soft ASP 3.6.2 for Windows. All configuration is performed during installation.

Before you begin, make sure you meet the following requirements:

- You are logged on to the server with administrator privileges.
- Your server configuration meets the requirements listed in "Installation Requirements: Sun Chili!Soft ASP for Windows" in this chapter.
- All applications that use ODBC connections are closed.
- The Web server is stopped. If you are running the Web server as an executable, stop the server by shutting it down or by closing the executable's window. If you are running the Web server as a service, stop the service via the Windows Control Panel.

Note

If you are upgrading from a previous version of Sun Chili!Soft ASP, you will run the setup program as described below, but first you must take additional steps. See "Upgrading Sun Chili!Soft ASP for Windows" in this chapter before beginning the upgrade installation.

To run the setup program and install Sun Chili!Soft ASP, use the following procedure.

To run the setup program

1. Launch the setup program:

   If you are installing Sun Chili!Soft ASP from a CD-ROM, insert the CD-ROM into the CD-ROM drive. If the setup program does not start automatically, double-click the setup.exe file located on the CD-ROM.

   If you are downloading Sun Chili!Soft ASP from the Web, download the installation file, casp362.exe, and then double-click the file.

2. Perform the step-by-step installation, as prompted by the setup program.

   Note: During installation you will be asked for the complete pathname of the folder in which you want to install Sun Chili!Soft ASP 3.6.2. If you are upgrading from a previous version of Sun Chili!Soft ASP, specify the folder in which the previous version is installed. Otherwise, you must uninstall the previous version.

   Also, to install and use the Sun Chili!Soft ASP ADO samples, Microsoft Data Access Components (MDAC) 2.5 or higher must be installed on your computer. If you opt to install the samples during the installation of Sun Chili!Soft ASP, and MDAC 2.5 is not yet installed, you have the option to install it. If you choose to install MDAC 2.5, the setup program walks

you through the MDAC installation. Following installation, restart your computer and then rerun the Sun Chili!Soft ASP 3.6.2 setup program as described in this topic.

3. After the installation of Sun Chili!Soft ASP is finished and you have restarted your computer, restart the Web server that was configured to run with Sun Chili!Soft ASP.

Important Notes

- Sun Chili!Soft ASP runs automatically whenever an ASP page is requested by a user (as long as the Web server is running). Sun Chili!Soft ASP runs until the Web server is stopped. When the Web server is restarted, Sun Chili!Soft ASP will not run until an ASP page is requested by a user.
- The Sun Chili!Soft ASP Administration Console is referenced throughout this documentation. The Administration Console is available for the UNIX and Linux versions of Sun Chili!Soft ASP only. With Sun Chili!Soft ASP for Windows, all configuration is performed during installation. Some of the Sun Chili!Soft ASP configuration information is stored in registry settings, however, and regedit32 can be used to edit those settings. For more information, see "Editing the Windows Registry" in "Chapter 3: Managing Sun Chili!Soft ASP."
- With Sun Chili!Soft ASP for Windows, ASP applications are defined by adding aliases or virtual directories to the Web server. Sun Chili!Soft ASP treats each alias or virtual directory as an ASP application. With Apache Web Server, ASP applications are defined by adding an alias to the httpd.conf file. With iPlanet Web Server, ASP applications are defined by adding an "additional document directory" using the server's Administration tool.
- ODBC drivers are not installed with Sun Chili!Soft ASP for Windows. Use the Windows ODBC Data Source Administrator (accessed from the Control Panel) to view installed ODBC drivers, and to create and manage DSNs.

See also:

Uninstalling Sun Chili!Soft ASP for Windows in this chapter

Upgrading Sun Chili!Soft ASP for Windows

If you are upgrading from a previous version of Sun Chili!Soft ASP, run the setup program as described in "Running the Setup Program" in this chapter. Before you begin, however, you must take the following steps:

- The new installation will completely overwrite your existing installation. If there are any items that you want to preserve in your current Sun Chili!Soft ASP installation directory, copy them to another location. Otherwise, they will be lost.
- If you changed your Sun Chili!Soft ASP registry settings from the defaults, you will need to reconfigure them following the upgrade (make note of your current registry settings before you begin the installation to use as a reference). For more information about configuring registry settings, see "Editing the Windows Registry" in "Chapter 3: Managing Sun Chili!Soft ASP."

- During installation you will be asked for the complete pathname of the folder in which you want to install Sun Chili!Soft ASP 3.6.2. Specify the folder in which the previous version is installed. Otherwise, you must uninstall the previous version.

Configuring the Web Server after Installation

This section describes how to configure a Web server to run with Sun Chili!Soft ASP 3.6.2, if you did not do so during installation. It also describes how to configure a non-DSO Apache Web Server after the installation of Sun Chili!Soft ASP. See "Configuring a Non-DSO Apache Web Server."

To configure a Web server after installation

1. Open the Administration Console by using the following URL:

   http://[HOSTNAME]:[PORT]

   where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

   The Administration Console **Add ASP Server** page displays a list of Web servers that have been detected on this computer.

2. Select the option button of the Web server you want to configure, and then go to step 4.

   - or –

   Click **Search Web servers** to refresh the list of detected Web servers, and then go to step 3.

   - or –

   Type the absolute path name of the configuration file for the Web server you want to configure, and then go to step 4.

3. The **Search Web servers** page displays, along with a **Web servers search** popup box. When the search is finished, the popup box displays the message **Done**. When you see this message, click **Add a Server** on the **Search Web servers** page, and then follow the instructions in step 2.

4. Select or deselect the **Insert Chili!Soft samples** check box to enable or disable the Sun Chili!Soft ASP sample applications, as desired, and then click **OK**.

   The Administration Console **Server Management** page appears. From this page, you can configure the ASP Server and Web server, as described in "Managing the ASP Server" and "Managing the Web Server" in "Chapter 3: Managing Sun Chili!Soft ASP."

Configuring a Non-DSO Apache Web Server

This topic applies if you have installed Sun Chili!Soft ASP on a computer running an Apache Web server that does not have Dynamic Shared Object (DSO) support.

When you install Sun Chili!Soft ASP on a computer running an Apache Web server, the setup program automatically links Sun Chili!Soft ASP to Apache through Apache's module facility. The setup program uses pre-built Sun Chili!Soft ASP DSO modules to set everything up for you. However, if you are running a non-DSO version of Apache (a version that does not have DSO support), you must use the following procedure to link Sun Chili!Soft ASP to the Apache Web server.

Note

The Sun Chili!Soft ASP setup program makes certain changes to the configuration files for the associated Web server. For more information about changes for Apache, see "Changes to Apache Web Server Configuration Files" in this chapter.

To manually link the Sun Chili!Soft ASP module to an Apache Web server that does not have DSO support, install Sun Chili!Soft ASP and then use the following procedure. The steps in this procedure are based on the assumption that Apache is installed and the source has been saved.

To link the Sun Chili!Soft ASP module to Apache Web Server

1. Stop the Apache Web server.
2. Copy the files "module/source/build/mod_casp2.c" and "module/source/build/dispint.h" from the Sun Chili!Soft ASP installation directory to the "src/modules/extra" subdirectory of your Apache Web server source tree directory.
3. From the Apache source tree directory, type the following (note that if you have a custom configuration of Apache, your settings might vary from this example):

   ```
   #> ./configure --prefix=[WEB_SERVER_ROOT_DIR] --activate-
   module=src/modules/extra/mod_casp2.c
   ```

   where `[WEB_SERVER_ROOT_DIR]` is the root directory for your installed Apache Web server.
4. (Important: On HP-UX, do not take this step; go to step 5 instead.) When the script has finished running, use a text editor to add "-ldl" to the EXTRA_LIBS section of src/Makefile.
5. Return to the Apache installation directory, and type the following:

   ```
   #> make
   ```
6. Copy the new Web server files to the appropriate location by typing:

   ```
   #> make install
   ```
7. Installation is automatic. When installation is complete, restart the Apache Web server, and then start the Sun Chili!Soft ASP Server, as described in "Stopping and Restarting the ASP Server" in "Chapter 3: Managing Chili!Soft ASP."

Changes to Web Server Configuration Files

When a Web server is configured to run with Sun Chili!Soft ASP (either during installation or after), the setup program makes certain changes to the iPlanet, Apache, or Zeus Web server configuration files. These changes are described in the following topics:

- Changes to iPlanet Web Server Configuration Files
- Changes to Apache Web Server Configuration Files
- Changes to Zeus Web Server Configuration Files

Changes to iPlanet Web Server Configuration Files

When Sun Chili!Soft ASP 3.6.2 is installed on a computer running iPlanet Web Server, the Sun Chili!Soft ASP setup program makes the following changes to the Web server's configuration files:

- It adds lines to the beginning of the obj.conf file (for Netscape Web Server 4.1) or the magnus.conf file (for iPlanet Web Server 6.0) as follows:

  On Solaris:

  In obj.conf:

  ```
  Init fn="load-modules" funcs="caspreq,caspinit,casptrans"
  shlib=
  "[C-
  ASP_INSTALL_DIR]/module/sunos5_optimized/netscape_6.x/nes_ca
  sp2.sl"
  Init fn="caspinit" casplib="[C-ASP_INSTALL_DIR]/asp-
  [server]-[PORT]"
  ```

  In magnus.conf:

  ```
  Init fn="load-modules" funcs="caspreq,caspinit,casptrans"
  shlib=
  "[C-
  ASP_INSTALL_DIR]/module/sunos5_optimized/netscape_6.x/nes_ca
  sp2.so"
  Init fn="caspinit" casplib="[C-ASP_INSTALL_DIR]/asp-
  [server]-[PORT]"
  ```

  On HP-UX:

  In obj.conf:

  ```
  Init fn="load-modules" funcs="caspreq,caspinit,casptrans"
  shlib=
  "[C-
  ASP_INSTALL_DIR]/module/ux11_optimized/netscape_6.x/nes_casp
  2.sl"
  ```

```
Init fn="caspinit" casplib="[C-ASP_INSTALL_DIR]/asp-
[server]-[PORT]"
```

In magnus.conf:

```
Init fn="load-modules" funcs="caspreq,caspinit,casptrans"
shlib=
"[C-
ASP_INSTALL_DIR]/module/ux11_optimized/netscape_6.x/nes_casp
2.so"
Init fn="caspinit" casplib="[C-ASP_INSTALL_DIR]/asp-
[Server]-[PORT]"
```

On Linux:

In obj.conf:

```
Init fn="load-modules" funcs="caspreq,caspinit,casptrans"
shlib=
"[C-
ASP_INSTALL_DIR]/module/linux2_optimized/netscape_6.x/nes_ca
sp2.sl"
Init fn="caspinit" casplib="[C-ASP_INSTALL_DIR]/asp-
[server]-[PORT]"
```

In magnus.conf:

```
Init fn="load-modules" funcs="caspreq,caspinit,casptrans"
shlib=
"[C-
ASP_INSTALL_DIR]/module/linux2_optimized/netscape_6.x/nes_ca
sp2.so"
Init fn="caspinit" casplib="[C-ASP_INSTALL_DIR]/asp-
[server]-[PORT]"
```

- It adds the following lines to the default object section of obj.conf:

```
<Object name=default>
NameTrans fn="casptrans"
Service method=(GET|POST) type="chilisoft-internal/active-
server-page" fn="caspreq" casplib="[C-ASP_INSTALL_DIR]/asp-
[server]-[PORT]"
</Object>
```

- It adds the following lines to the mime.types file:

```
type=chilisoft-internal/active-server-page exts=asp,asa
[C-ASP_INSTALL_DIR] resembles: /opt/casp
```

```
[SERVER] resembles: netscape

[PORT] resembles: 3000
```

- It comments out support for ASAP WebShow in the mime.types file because it also uses the .asp extension:

```
## by Chili!Soft ASP install: type=application/x-asp
exts=asp
```

Note

Sun Chili!Soft ASP supports only one instance of **LoadObjects** in the iPlanet Web Server magnus.conf file.

Changes to Apache Web Server Configuration Files

When Sun Chili!Soft ASP 3.6.2 is installed on a computer running Apache Web Server, the Sun Chili!Soft ASP setup program makes the following changes to the Web server configuration file (httpd.conf):

- It adds these lines:

```
AddHandler chiliasp .asp

AddHandler chiliasp .asa

CaspLib [C-ASP_INSTALL_DIR]/asp-[server]-[PORT]
```

- It adds these lines:

  On Solaris:

```
LoadModule  casp2_module

[C-
ASP_INSTALL_DIR]/module/sunos5_optimized/apache_[VERSION]/

[API]/mod_casp2.so
```

  On HP-UX:

```
LoadModule  casp2_module

[C-ASP_INSTALL_DIR]/module/ux11_optimized/apache_[VERSION]/

[API]/mod_casp2.so
```

  On Linux:

```
LoadModule  casp2_module

[C-
ASP_INSTALL_DIR]/module/linux2_optimized/apache_[VERSION]/

[API]/mod_casp2.so
```

- It adds this line:

```
MaxRequestsPerChild  30000
```

Changes to Zeus Web Server Configuration Files

When Sun Chili!Soft ASP 3.6.2 is installed on a computer running Zeus Web Server, the Sun Chili!Soft ASP setup program makes the following changes to the Web server's configuration files:

- It adds lines to the beginning of the obj.conf file as follows:

  On Solaris:

  ```
  Init fn="load-modules" funcs="caspreq,caspinit,casptrans"
  shlib=
  "[C-
  ASP_INSTALL_DIR]/module/sunos5_optimized/zeus/zeus_nsapi_cas
  p2.so"
  Init fn="caspinit" casplib="[C-ASP_INSTALL_DIR]/asp-
  [server]-[PORT]"
  ```

  On HP-UX:

  ```
  Init fn="load-modules" funcs="caspreq,caspinit,casptrans"
  shlib=
  "[C-
  ASP_INSTALL_DIR]/module/ux11_optimized/zeus/zeus_nsapi_casp2
  .so"
  Init fn="caspinit" casplib="[C-ASP_INSTALL_DIR]/asp-
  [server]-[PORT]"
  ```

  On Linux:

  ```
  Init fn="load-modules" funcs="caspreq,caspinit,casptrans"
  shlib=
  "[C-
  ASP_INSTALL_DIR]/module/linux2_optimized/zeus/zeus_nsapi_cas
  p2.so"
  Init fn="caspinit" casplib="[C-ASP_INSTALL_DIR]/asp-
  [server]-[PORT]"
  ```

- It adds the following lines to the default object section of obj.conf:

  ```
  <Object name="default">
  NameTrans fn="casptrans"
  Service method=(GET|POST) type="chilisoft-internal/active-
  server-page" fn="caspreq" casplib="[C-ASP_INSTALL_DIR]/asp-
  [server]-[PORT]"
  ```

```
</Object>
```

- It adds the following lines to the mime.types file:

```
type=chilisoft-internal/active-server-page exts=asp,asa
[C-ASP_INSTALL_DIR] resembles: /opt/casp
[SERVER] resembles: Zeus
[PORT] resembles: 3000
```

Note

Sun Chili!Soft ASP only supports one instance of **LoadObjects** in the Zeus Web Server magnus.conf file.

Changing Installation Options after Installation

During the installation of Sun Chili!Soft ASP you have many options regarding ASP Server and Web server configuration. Some of the choices you make during installation can be changed after installation. You can change the:

- Web server with which Sun Chili!Soft ASP is configured to run.
- Status of the "start on system boot" functionality (enabled or disabled), which determines if Sun Chili!Soft ASP starts automatically each time the computer is started.
- Status of Java support (enabled or disabled). To use Chili!Beans, Java support must be enabled (a Java runtime environment must be installed).

In this section:

- Changing the Web Server after Installation
- Enabling Java Support
- Starting Sun Chili!Soft ASP on System Boot

Changing the Web Server after Installation

In certain cases you might want to change the Web server with which Sun Chili!Soft ASP is configured to run. This association is referred to as the Web server-to-ASP Server linkage, and was specified during the installation of Sun Chili!Soft ASP.

To change the Web server after installation

1. From the Sun Chili!Soft ASP installation directory (/opt/casp by default), type the following command:

   ```
   #> ./configure-server
   ```

2. At the prompt, select **Configure the ASP Server**.

3. Select **Change the Web server-to-ASP Server association**.
4. At the prompt, select the Web server you want to change, or select **Cancel** to exit.

   Note: If no Web servers are installed, you will be prompted to add a server.
5. At the prompt, enter **y** (yes) if the ASP Server information is correct.

   - or -

   If the information is incorrect, enter **n** (no) to return to the previous screen.
6. At the prompt, select a Web server from the list, or make another selection:

   Select **Specify the Web server** to specify the Web server manually, and then make your selections as prompted.

   - or -

   Select **Attempt to auto-detect more Web servers** to direct the system to search for (auto-detect) installed Web servers from which to select, and then make your selection.

   - or -

   Select **Do not configure a Web server** to cancel the operation altogether. Choosing this option returns you to step 2.
7. At the prompt on the **Verify Web Server Information** screen, enter **y** (yes) if the Web server information is correct.

   Note: If the information is incorrect, enter **n** (no) to return to the previous screen.
8. At the prompt, select the desired configuration option:

   Choose **1. Default configuration** to use the default configuration settings and finish the reconfiguration of the Web server. This option is strongly recommended for all but the most experienced users of Sun Chili!Soft ASP.

   - or -

   Choose **2. Custom configuration** if you are an experienced user of Sun Chili!Soft ASP and want to customize a number of settings. If you select this option, you will also receive a prompt asking you if you want the Web server restarted. If you enter **y** (yes), the Web server will be restarted and configured. If you enter **n** (no), you will be prompted to restart the Web server manually.

   - or -

   Choose **3. Choose another Web server to install to** if you do not want to reconfigure the Sun Chili!Soft ASP Server-to-Web server association. Choosing this option returns you to step 4.

Caution

> If you select the first or second option, any current Sun Chili!Soft ASP Server-to-Web server association will be lost, disabling the previously associated Web server from serving up ASP content. If you do not want to reconfigure this association, choose the third option (to choose another Web server).

Note

If reconfiguration fails for any reason, the current association is left unchanged.

Enabling Java Support

If your ASP applications make use of Chili!Beans, a Java runtime environment (JRE) must be installed on the machine, and Chili!Beans must be enabled from the Sun Chili!Soft ASP Administration Console. If you did not install a JRE during installation of Sun Chili!Soft ASP, you can do so by using the following procedure.

To enable or disable Java support

1. From the Sun Chili!Soft ASP installation directory (/opt/casp by default), type the following command:

   ```
   #> ./configure-server
   ```

2. At the prompt, make the desired selection:

   Select **Configure Java support** to continue.

   – or –

   Select **Exit without performing any action** to exit.

3. On the screen that pertains to configuring Chili!Bean support, enable or disable Java support as desired:

   Select **Use the bundled 1.3.1 JRE** to install JRE 1.3.1 (JRE 1.3.1 is bundled with this version of Sun Chili!Soft ASP and is the recommended JRE). Then, respond to the prompts.

   – or –

   Select **Specify the path to an existing JRE** to specify a different JRE (versions 1.2.x and 1.3.x are supported) or type **none** to return to the previous screen. Then, respond to the prompts.

   – or –

   Select **Disable Java support**, and then respond to the prompts.

   – or –

   Select **Keep your current settings**.

See also:

Enabling Chili!Beans in "Chapter 5: Developer’s Reference"

Chili!Beans Component Reference in "Chapter 5: Developer’s Reference"

Starting Sun Chili!Soft ASP on System Boot

The default installation of Sun Chili!Soft ASP is to automatically start the ASP Server on system boot. This option was configured during installation. To enable or disable the "start on system boot" functionality, use the following procedure.

To start Sun Chili!Soft ASP on system boot

1. From the Sun Chili!Soft ASP installation directory (/opt/casp by default), type the following command:

   ```
   #> ./configure-server
   ```

2. At the prompt, select **Configure the ASP Server**.
3. Select **Enable or disable the 'start on system boot functionality'**, and then make your desired selection. If this option is enabled, Sun Chili!Soft ASP starts automatically each time the computer is started.

Uninstalling Sun Chili!Soft ASP

This section includes the following topics about uninstalling Sun Chili!Soft ASP:

- Uninstalling Sun Chili!Soft ASP for UNIX and Linux
- Uninstalling Sun Chili!Soft ASP for Windows

Uninstalling Sun Chili!Soft ASP for UNIX and Linux

On UNIX and Linux systems, Sun Chili!Soft ASP is uninstalled by running the script named `uninstall`, which is located in the Sun Chili!Soft ASP installation directory.

When you run the uninstall program, you can delete all directories and files contained in the Sun Chili!Soft ASP installation directory. Before you run the uninstall program, make copies of any files contained under this directory that you do not want to lose.

Note

You must be logged in as root on the computer running Sun Chili!Soft ASP.

To uninstall Sun Chili!Soft ASP for UNIX and Linux

1. From the Sun Chili!Soft ASP installation directory (/opt/casp by default), type the following command:

   ```
   #> ./uninstall
   ```

2. Choose the number of the desired uninstall method. Choose **1. Uninstall the entire product** to remove all Sun Chili!Soft ASP components and all files and directories under the installation directory. If you choose this option, go to step 5.

   – or –

Choose **2. Perform a stage-based uninstall** to select the components to uninstall. If you choose this option, go to step 3.

– or –

Choose **3. Cancel the uninstall** to stop the uninstall without removing any files.

3. If you chose option **2. Perform a stage-based uninstall**, you are prompted to select the components to uninstall. At the prompts, enter the number of the component(s) you want removed.
4. Uninstall the Web server-Sun Chili!Soft ASP association, responding to the prompts as desired.
5. Delete the directories and files in the Sun Chili!Soft ASP installation directory. Before deletion you are prompted to stop the uninstall and make backup copies of any files. To perform this action, enter **y** (yes). To continue the uninstall and delete the directories and files, enter **n** (no).

Uninstalling Sun Chili!Soft ASP for Windows

To uninstall Sun Chili!Soft ASP on Windows systems, use the following procedure.

Note

To perform this procedure, you must have administrator rights.

To uninstall Sun Chili!Soft ASP for Windows

1. Log on to the computer running Sun Chili!Soft ASP.
2. Stop the Web server that is configured to run with Sun Chili!Soft ASP.
3. On the Windows **Start** menu, point to **Programs**, and then point to **Sun Chili!Soft ASP**.
4. Click **Uninstall Sun Chili!Soft ASP**.
5. On the **Uninstalling** screen, click **Next** to begin uninstalling Sun Chili!Soft ASP.
6. On the **Uninstall Complete** screen, click **Finish**.

Enabling Publishing

With Sun Chili!Soft ASP, Web developers and authors can publish their work to the Web server in the usual manner, such as by using FTP. In addition, Sun Chili!Soft ASP supports Microsoft FrontPage Server Extensions, so that users of Microsoft FrontPage can publish Web pages and applications to the Web server by using the FrontPage client. For more information about installing FrontPage Server Extensions and setting up FrontPage users, see "Enabling FrontPage Publishing" in "Chapter 3: Managing Sun Chili!Soft ASP."

Note

FrontPage Server Extensions are not installed with Sun Chili!Soft ASP; you must obtain them from Microsoft. Specific questions about the installation, configuration, and use of FrontPage and FrontPage Server Extensions should be directed to Microsoft or its representatives.

Defining ASP Applications on the Server

Sun Chili!Soft ASP includes the concept of an ASP application, which comprises a hierarchical set of directories that contain the ASP pages and other files used by the application. The root directory of an ASP application contains an optional global.asa file, which stores application state information along with Application and Session information. Using the **Application** and **Session** objects with the global.asa file is explained in "Using the global.asa File" in "Chapter 4: Building a Sun Chili!Soft ASP Application."

For an ASP application to be processed, it must be defined on the ASP Server. There are several ways to define an application on Sun Chili!Soft ASP:

- Add the application from the Administration Console, as described in "Adding an ASP Application" in "Chapter 3: Managing Sun Chili!Soft ASP."
- Enable ASP processing for a Virtual Host, as described in "Enabling ASP for a Virtual Host" and "Defining Applications in a Shared Environment" in "Chapter 3: Managing Sun Chili!Soft ASP."
- Use the FrontPage Services file, as described in "Using the Frontpage Services File in a Shared Environment" in "Chapter 3: Managing Sun Chili!Soft ASP."
- Add an application from within the Sun Chili!Soft ASP configuration file, or add an alias from within the Apache Web server configuration file. This is an advanced administration option described in "Defining Applications on UNIX" in "Chapter 3: Managing Sun Chili!Soft ASP."

Note

On Windows NT and Windows 2000, ASP applications are defined by adding aliases or virtual directories to the Web server. Sun Chili!Soft ASP treats each alias or virtual directory as an ASP application. With Apache Web Server, ASP applications are defined by adding an alias to the httpd.conf file. With iPlanet Web Server, ASP applications are defined by adding an "additional document directory" using the Web server's Administration tool.

You can access sample ASP applications from the Administration Console, as described in "Accessing Documentation, Samples, and Diagnostics" in "Introduction: About This Documentation."

Enabling Database Connections on the Server

With Sun Chili!Soft ASP, you can easily display and manipulate information stored in a database from an ASP page. To enable an ASP application to retrieve data from a database, the system administrator must first configure the Sun Chili!Soft ASP Server to connect to the database. Then the Web developer can create and initialize a connection to the database from within the application. This topic provides overview information about enabling a connection on the ASP Server. For more detailed instructions, see "Configuring a Database" in "Chapter 3: Managing Sun Chili!Soft ASP."

Sun Chili!Soft ASP provides built-in ActiveX Data Object (ADO) control that developers can use from within an ASP application to initialize a database connection for retrieving and manipulating data. ADO provides the interface through which ODBC drivers are called and provides "containers" for storing information that is passed to and from the database. The most common container is a **Recordset** object, which stores the results of a **SELECT** SQL query. The ADO **Connection** object establishes connections to databases by using ODBC drivers. For more information about ADO, see "ADO Component Reference" in "Chapter 5: Developer's Reference."

For UNIX and Linux versions of Sun Chili!Soft ASP, the setup program automatically installs ODBC drivers for a number of different databases (Sun Chili!Soft ASP for Windows does not install ODBC drivers). You can view the list of installed drivers from the Administration Console, as described in "Viewing the List of ODBC Drivers" in "Chapter 3: Managing Sun Chili!Soft ASP." For Windows systems, the list of installed ODBC drivers can be viewed from the Windows Control Panel (double-click the **Data Sources** icon). See Microsoft documentation for more information about using the Windows ODBC Data Source Administrator.

Sun Chili!Soft ASP includes DataDirect SequeLink 4.51a, which enables connections to remote computers running Microsoft Access or Microsoft SQL Server 6.5. For more information, see "Configuring SequeLink" in "Chapter 3: Managing Sun Chili!Soft ASP."

ADO and either the appropriate ODBC driver or SequeLink are required to create a connection to a particular database. ADO uses connection information and the ODBC driver manager to create an instance of the required ODBC driver, which in turn connects to the database.

With Sun Chili!Soft ASP, Web developers can specify the connection information for the database by using system DSNs, file DSNs, or DSN-less connection strings. The appropriate method to use depends on user preferences and the environment in which Sun Chili!Soft ASP is running. For more information, see "Connecting to a Database" in "Chapter 4: Building a Sun Chili!Soft ASP Application."

For enterprise applications, it is recommended that ASP developers use system DSNs. The system administrator can use the Sun Chili!Soft ASP Administration Console to create system DSNs, which can be referenced from within an ASP application for initializing the database connection. For more information about creating a system DSN, see "Configuring Data Source Names (DSNs)" in "Chapter 3: Managing Sun Chili!Soft ASP."

In a shared Web hosting environment, such as with an Internet Service Provider (ISP), using system DSNs poses two problems:

- A DSN that includes a username and password for the database makes the data source accessible from any ASP page on the server, representing a security risk.
- Creating DSNs for each customer can be a significant administrative burden for the Web hosting provider. Because Web developers can create them and the database username and password information can be restricted to a specific ASP application, using file DSNs and DSN-less connection strings are more appropriate in a Web hosting environment.

Note

It is strongly recommended that you validate your database connection parameters prior to creating a database connection with Sun Chili!Soft ASP. An ODBC driver can bring down your ASP Server if it is passing incorrect parameters. You should test your database connections on a nonproduction server.

The following example illustrates the relationship between Sun Chili!Soft ASP, ADO, ODBC drivers, and databases.

A connection string on the ASP page specifies the information required by both ADO and the ODBC driver manager for connecting to the database. The following example uses a DSN-less connection string:

```
connect_string = "Driver={ODBC_driver_name};
Database=[database_name]; UID=[username]; PWD=[password]"
```

The next line of code creates an instance of the ADO **Connection** object:

```
set dbConn = server.createObject ("ADODB.connection")
```

The following code calls the **Open** method of the ADO **Connection** object, which takes the **connection_string** parameter. In this step, ADO requests that the ODBC driver manager create an instance of the specified ODBC driver. ADO passes the remainder of the connection string to the ODBC driver, which uses this information to connect to the database.

```
open dbConn connect_string
```

Chapter 3: Managing Sun Chili!Soft ASP

This chapter provides detailed information about managing Sun Chili!Soft ASP, including information about changing configuration settings, enhancing security, and optimizing server performance. It describes how to use the Sun Chili!Soft ASP Administration Console, which provides browser-based access to Sun Chili!Soft ASP configuration settings. For advanced users, "Advanced Administration Options" describes how to manage Sun Chili!Soft ASP by editing configuration files directly or by using provided scripts. However, because you can create serious problems with your system in this manner, it is strongly recommended that you use the Administration Console for configuration tasks whenever possible.

Note

> The Sun Chili!Soft ASP Administration Console is not available with Sun Chili!Soft ASP for Windows. On Windows systems, regedit32 can be used to edit some of the Sun Chili!Soft ASP configuration settings. For more information, see "Editing the Windows Registry" in this chapter.

Who should read this chapter: System administrators responsible for configuring and running Sun Chili!Soft ASP.

In this chapter:

- Using the Administration Console
- Managing the ASP Server
- Managing the Web Server
- Enabling FrontPage Publishing
- Configuring a Database
- Configuring ActiveX Data Objects (ADO) Connections
- Running Sun Chili!Soft ASP in a Web Hosting Environment
- Optimizing Server Performance
- Advanced Administration Options

Using the Administration Console

Most configuration settings for Sun Chili!Soft ASP are accessible from the Sun Chili!Soft ASP Administration Console. The Administration Console is a browser-based application used for managing Sun Chili!Soft ASP.

Note

The Administration Console is not available with the Windows version of Sun Chili!Soft ASP. On Windows NT and Windows 2000, regedit32 can be used to edit some of the Sun Chili!Soft ASP configuration settings. For more information, see "Editing the Windows Registry" in this chapter.

The topics in this section explain how to access, configure, and use the Administration Console. This section covers:

- Accessing the Administration Console
- Starting and Stopping the Administration Web Server
- Configuring Usernames and Passwords
- Accessing the Product Documentation
- Contacting Customer Support
- Installing a New Serial Number
- Checking for Product Updates

Other sections in this chapter explain how to use the Administration Console to configure and manage Sun Chili!Soft ASP:

- Managing the ASP Server
- Managing the Web Server
- Enabling FrontPage Publishing
- Configuring a Database
- Configuring ActiveX Data Objects (ADO) Connections
- Optimizing Server Performance

Accessing the Administration Console

The Sun Chili!Soft ASP Administration Console enables you to configure and manage a Sun Chili!Soft ASP Server from a Web browser, either locally or remotely. The Administration Console also enables you to start and stop the Web server that is configured to run with Sun Chili!Soft ASP.

The Administration Console is hosted by the Administration Web site, which is installed on the computer running the ASP Server. The Administration Web site consists of its own Apache Web Server and its own ASP Server. By default, the Administration Web site is configured to start when you start the computer running Sun Chili!Soft ASP.

Note

If you did not configure a Web server to run with Sun Chili!Soft ASP during installation, you will be prompted to configure one the first time you open the Administration Console. On Cobalt platforms, a Web server is configured automatically during installation.

Note

On Cobalt platforms, you can also access the Sun Chili!Soft ASP Administration Console from the Cobalt Administration Console.

To access the Administration Console

1. In your browser address bar, type the following URL:

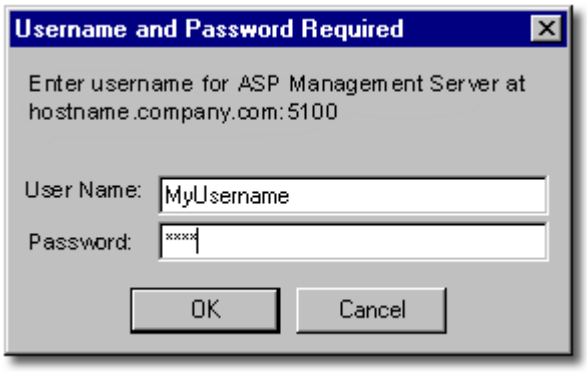
   http://[HOSTNAME]:[PORT]

   where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default; for more information, see the note below).

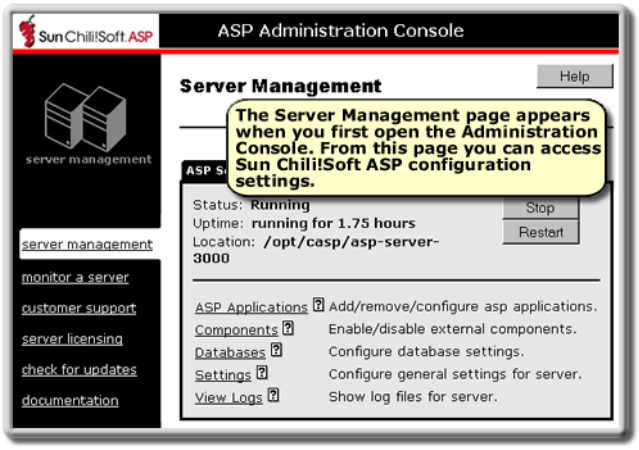
   For example, if your [HOSTNAME] is www.mysite.com and the Administration Console is installed to port 5100, you would access the Administration Console at:

   http:// www.mysite.com:5100/

2. When prompted, type the username and password specified during installation. If you are running a Cobalt system, or if you chose a default configuration for the Administration Console during installation, the setup program automatically sets the username to "admin" and the password to "root." You should change these from the defaults as soon as possible. You can change the username and password by following the instructions in "Configuring Usernames and Passwords" in this chapter.



The Administration Console opens, displaying the **Server Management** page.



From the **Server Management** page you can access the configuration settings for Sun Chili!Soft ASP, view information about your installation, and start and stop the associated Web server.

Note

When Sun Chili!Soft ASP is installed and your Web server is running, you can also access the Administration Console from the Sun Chili!Soft ASP Start Page at:

http://[HOSTNAME]/caspsamp/

where [HOSTNAME] is the hostname of your Web server.

To access the Sun Chili!Soft ASP Administration Console by using a URL, you must specify its port number in the URL. The default port number is 5100. However, it might be a different number if 5100 was already in use when you installed Sun Chili!Soft ASP, or if you specified a different port number for the Administration Web site during installation. If you are unsure of the correct port number, you can find this information in the Sun Chili!Soft ASP installation summary file. This file is named install_summary on Solaris, AIX, and UP-UX, and component_log on Linux and Cobalt RaQ3.

On UNIX and Linux platforms, you can find the installation summary file in the following location:

/[C-ASP_INSTALL_DIR]/logs/

where [C-ASP_INSTALL_DIR] is the directory in which you installed Sun Chili!Soft ASP.

On Cobalt platforms, you can find the installation summary file in the following location:

/home/chiliasp/logs/

The entry reads as follows:

```
Administration console installed:

URL: http://[WEB_SERVER_HOSTNAME]:[PORT_NUMBER]

Port: [PORT_NUMBER]
```

See also:

Using the Administration Console in this chapter

Starting and Stopping the Administration Web Server

When you install Sun Chili!Soft ASP, the setup program installs an administration Web server, which hosts the Sun Chili!Soft ASP Administration Console. By default, the administration Web server is configured to start when you start the computer running Sun Chili!Soft ASP. Although you can start and stop the ASP Server by using the Administration Console (as described in "Stopping and Restarting the ASP Server" in this chapter), to start or stop the administration Web server, you must use the command-line utility, admtool, which is installed with Sun Chili!Soft ASP.

To start or stop the administration Web server

1. Telnet or log in to the computer running Sun Chili!Soft ASP as root.
2. Change directories (cd) to the Sun Chili!Soft root installation directory (/opt/casp by default).
3. Start the admtool utility by running the following command:

   ```
   ./admtool
   ```

   When you start the admtool utility, the following list of options displays:

   1. Start admin server. Starts the administration Web server.

   2. Stop admin server. Stops the administration Web server.

   3. Admin server status. Indicates whether the administration Web server is running or stopped.

   4. Add a user. Adds a new administrator username and password or changes the password for an existing username.

   5. Remove a user. Removes a username.

   6. List users. Shows a list of all usernames currently configured for the Administration Console.

   7. Quit. Saves any changes and exits the admtool utility.
4. To start the administration Web server, enter **1** (**Start admin server**)

   – or –

   To stop the administration Web, enter **2** (**Stop admin server**).
5. When prompted, press **Enter** to continue.
6. To save any changes and exit, enter **7** (**Quit)**.

See also:

Accessing the Administration Console in this chapter

Stopping and Restarting the ASP Server in this chapter

Starting and Stopping the Web Server in this chapter

Configuring Usernames and Passwords

During installation, the Sun Chili!Soft ASP setup program creates a username and password to restrict access to the Sun Chili!Soft ASP Administration Console. You can add, edit, and delete usernames and passwords by using the admtool utility, which is installed with Sun Chili!Soft ASP. You cannot configure usernames and passwords for the Administration Console from within the Administration Console.

Important Security Note

> During installation, if you are running a Cobalt system or if you chose a default configuration for the Administration Console, the setup program sets the administrator username to "admin" and the default password to "root." To protect the security of your server, change the username and password from the defaults as soon as possible.

Sun Chili!Soft ASP enables you to configure usernames and passwords for one or more Administration Console users. To configure usernames and passwords, use the following procedure.

To configure usernames and passwords

1. Telnet or log in to the computer running Sun Chili!Soft ASP as root.
2. Change directories (cd) to the Sun Chili!Soft ASP root installation directory (/opt/casp by default).
3. Start the admtool utility by using the following command:

   ```
   ./admtool
   ```

4. When you start the admtool utility, the following list of options displays. Enter the number of the option you want (**4** to add or change a username and password, **5** to remove a username, or **6** to see the current list of usernames), and then follow the prompts.

   1. Start admin server. Starts the administration Web server.

   2. Stop admin server. Stops the administration Web server.

   3. Admin server status. Indicates whether the administration Web server is running or stopped.

   4. Add a user. Adds a new administrator username and password or changes the password for an existing username. To add a username and password, enter the new username and password when prompted. To change the password for an existing username, enter the username and the new password when prompted.

   5. Remove a user. Removes a username. At the prompt, enter the username you want to remove.

6. List users. Displays a list of all usernames currently configured for the Administration Console.

7. Quit. Saves any changes and exits the admtool utility.

5. When prompted, press **Enter** to continue.
6. To continue configuring more usernames and passwords, enter the number of the option you want.
7. When finished, enter **7** (**Quit**) to save your changes and exit the admtool utility.

See also:

Accessing the Administration Console in this chapter

Accessing the Product Documentation

The Sun Chili!Soft ASP setup program installs two versions of the product documentation: one in HTML format that includes dynamic index and search functionality, and one in Adobe PDF format. There are three primary ways to access the documentation:

- If you chose the option to enable documentation during installation, you can access the HTML version of the documentation by using the following URL:

  http://[HOSTNAME]/caspdoc/

  where [HOSTNAME] is the hostname of your Web server.

  From the first page of the HTML documentation, click a link to open the version in Adobe PDF format. To access the Adobe PDF version directly, use the following URL:

  http://[HOSTNAME]/caspsamp/pdf

  where [HOSTNAME] is the hostname of your Web server.

  To view and print the Adobe PDF version, you must have Adobe Acrobat Reader installed. To obtain a free copy, go to:

  http://www.adobe.com/products/acrobat/readstep2.html

- You can access both versions of the product documentation on the Sun Chili!Soft Web site at:

  http://www.chilisoft.com/caspdoc/

- You can access the documentation from the Sun Chili!Soft ASP Administration Console using the following procedure.

To access the product documentation from the Administration Console

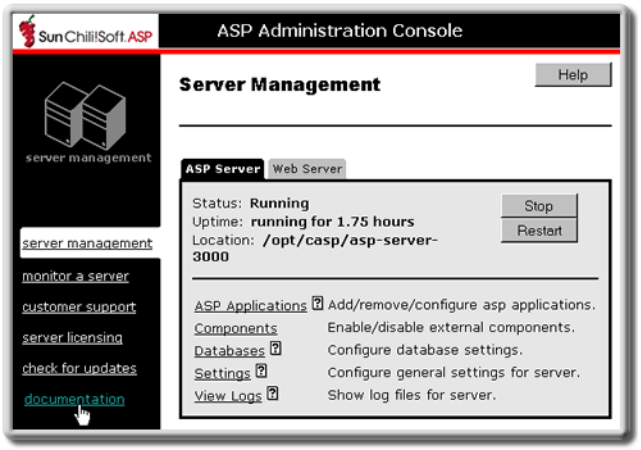
1. Open the Administration Console by using the following URL:

   http://[HOSTNAME]:[PORT]

where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

The **Server Management** page displays.

2. In the left navigation pane, click **documentation**.



Viewing the Product README File

When you install Sun Chili!Soft ASP a README file is installed on your computer. The README provides the most current product information. You can access the README file from the Sun Chili!Soft ASP Administration Console, as described later in this topic. You can also find the README file in the following directory:

/[C-ASP_INSTALL_DIR]/

where [C-ASP_INSTALL_DIR] is the path name of the Sun Chili!Soft ASP installation directory (/opt/casp by default on UNIX and Linux, and /home/chiliasp on RaQ3).

Note

A README file is not provided for Cobalt RaQ4 and Cobalt XTR.

In addition to the README file, Sun Chili!Soft provides a number of other resources to answer your questions about configuring and using Sun Chili!Soft ASP. These resources are described in "Introduction: About This Documentation."

To view the product README file

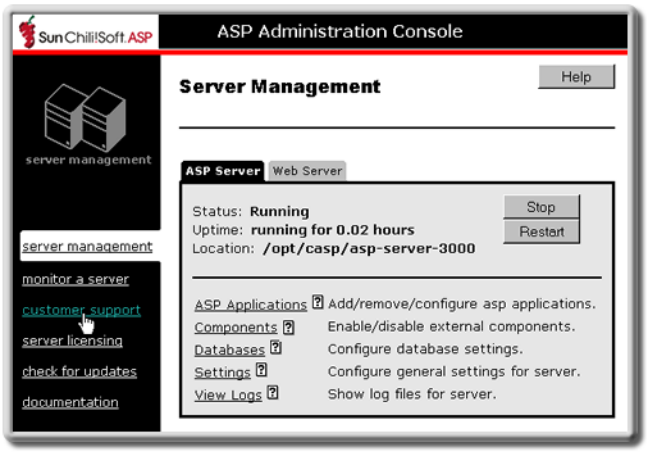
1. Open the Administration Console by using the following URL:

   http://[HOSTNAME]:[PORT]

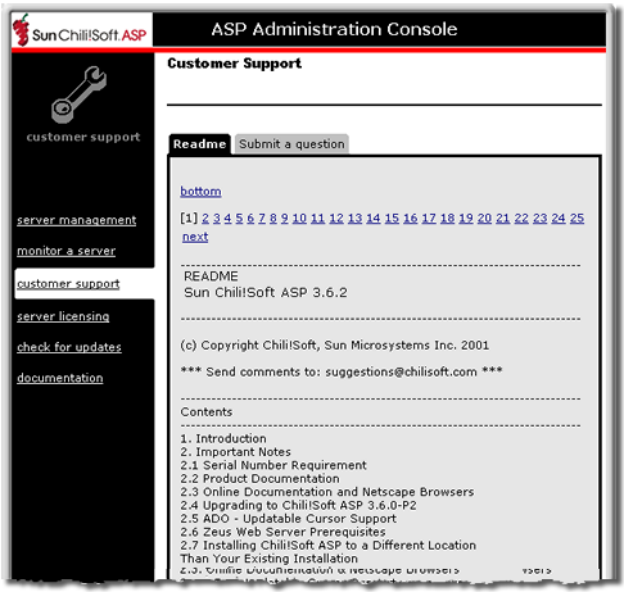
   where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

   The **Server Management** page displays.

2. In the left navigation pane, click **customer support**.



The README file displays.



Contacting Customer Support

If you encounter a problem while using Sun Chili!Soft ASP, you can use the following procedure to contact Sun Chili!Soft Customer Support from the Sun Chili!Soft ASP Administration Console **Customer Support** page.

Note

This page is not available for Cobalt RaQ4 and Cobalt XTR. To contact Customer Support, see step 5 below.

To contact Customer Support

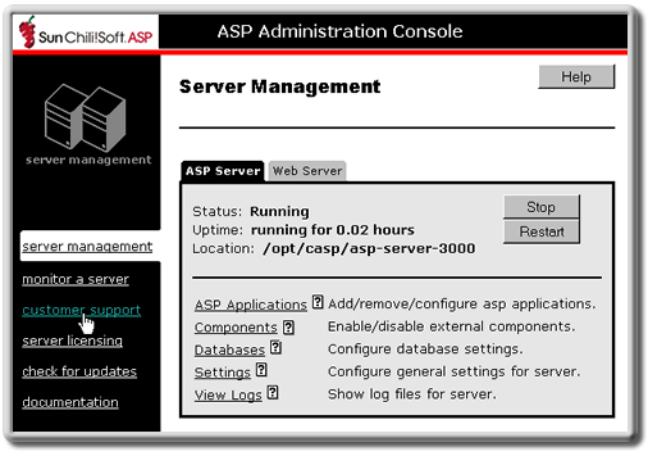
1. Open the Administration Console by using the following URL:

   http://[HOSTNAME]:[PORT]

   where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

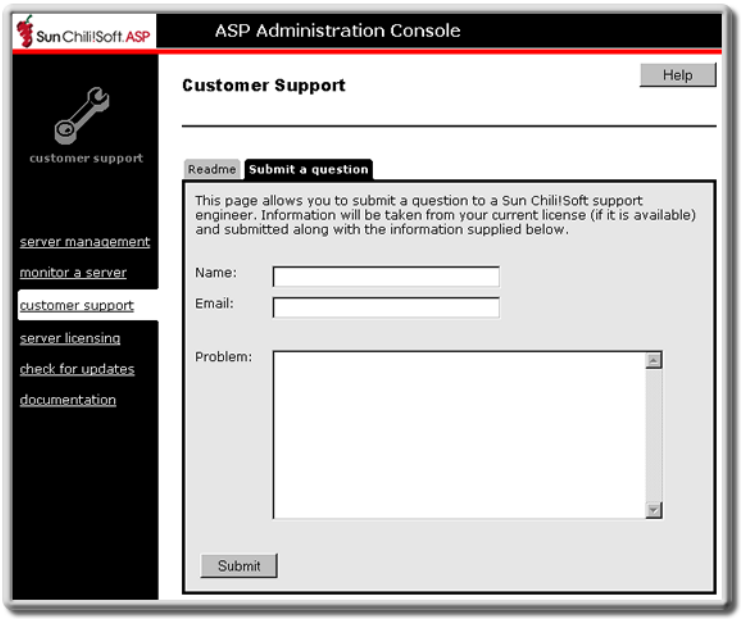
   The **Server Management** page displays.

2. In the left navigation pane, click **customer support**.



3. On the **Customer Support** page, click the **Submit a question** tab.

   The **Submit a question** page displays.



4. In the text boxes, type your name, e-mail address, and a description of the problem.
5. Click **Submit**.

If you are unable to submit your problem as described in the previous steps, contact the Sun Chili!Soft Customer Support team using the Web form at the following address:

http://www.chilisoft.com/support/chili.eval.asp

If you do this, be sure to include the license number that is displayed on the **ASP Server Licensing** page. To view this page, click **server licensing** in the left navigation pane of the Sun Chili!Soft ASP Administration Console.

Installing a New Serial Number

When you upgrade your product license (for example, if you're upgrading from an evaluation version of Sun Chili!Soft ASP to a full version), you must install a new serial number that you receive from Sun Chili!Soft. The Sun Chili!Soft ASP Administration Console **ASP Server Licensing** page displays information about your current Sun Chili!Soft ASP license, and enables you to install a new serial number.

To install a new Sun Chili!Soft ASP serial number

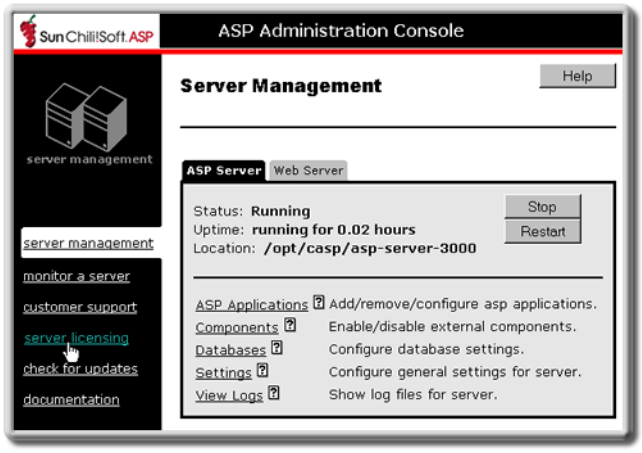
1. Open the Administration Console by using the following URL:

   http://[HOSTNAME]:[PORT]

   where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

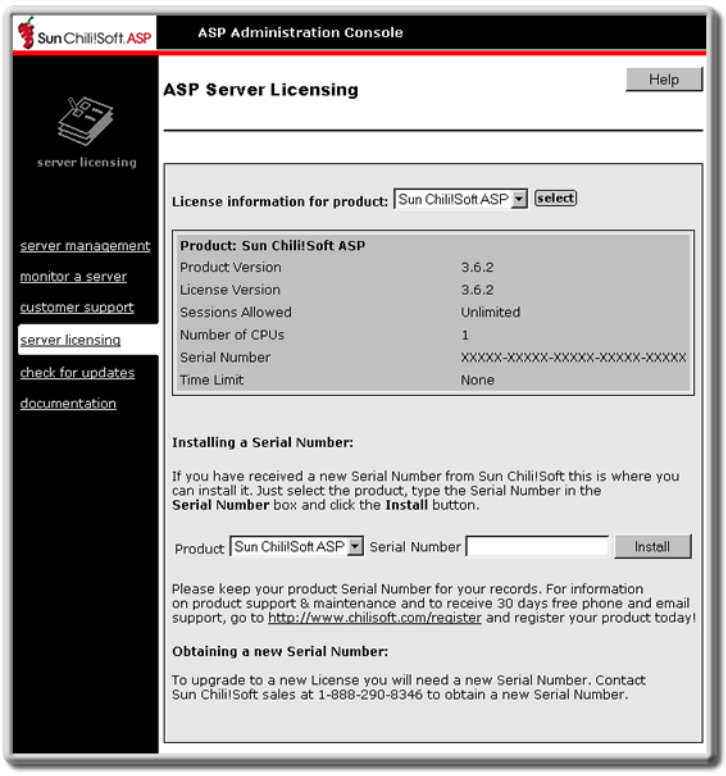
   The **Server Management** page displays.

2. In the left navigation pane, click **server licensing**.



   The **ASP Server Licensing** page displays with license information.

3. In the **Product** drop-down list, select **Sun Chili!Soft ASP**.



4. In the **Serial Number** box, type the new serial number, and then click **Install**.
5. To put your changes into effect, restart the ASP Server by clicking **Restart**.

Note

Restarting the ASP Server resets all **Session** and **Application** variables.

Checking for Product Updates

Sun Chili!Soft occasionally provides product updates and fixes to enhance the security and performance of Sun Chili!Soft ASP software. When using the Administration Console you will see periodic prompts asking you if you want to check for updates. You can also check for updates on demand, to quickly determine if your specific installation of Sun Chili!Soft ASP is up-to-date. To check for updates on demand from the Administration Console, use the following procedure.

Note

Configuration information transmitted while checking for updates DOES NOT contain any personal or company identifying information.

This feature is not available with Sun Chili!Soft ASP for Windows.

To check for product updates

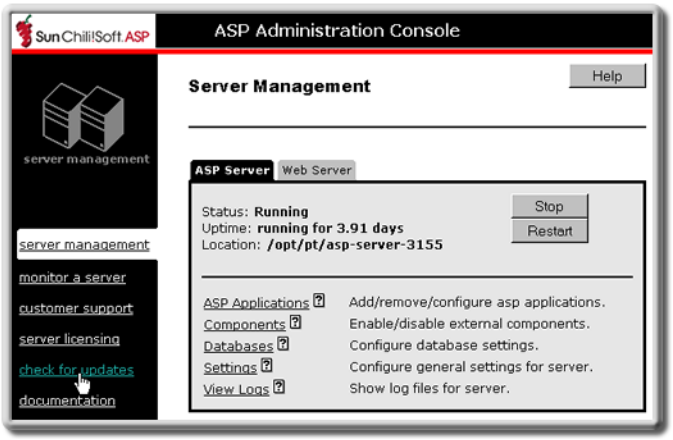
1. Open the Administration Console by using the following URL:

http://[HOSTNAME]:[PORT]

where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

The **Server Management** page displays.

2. In the left navigation pane, click **check for updates**.



The **Sun Chili!Soft Product Update** page displays, listing information about your Sun Chili!Soft ASP installation.

3. Make your desired selection:

Select **Check for update now** to check the Sun Chili!Soft Web site for updates and transmit your installation information.

- or -

Select **Do not ask for 90 days** to be prompted to check for updates again in 90 days.

Managing the ASP Server

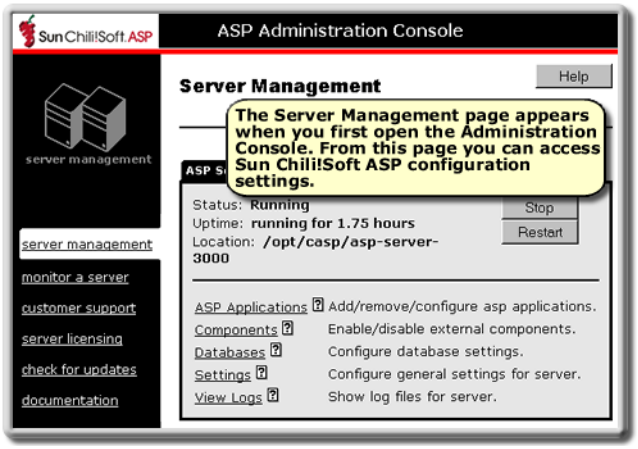
Sun Chili!Soft ASP includes an ASP Server that processes ASP page requests. On UNIX and Linux systems, the ASP Server is managed from the **Server Management** page in the Administration Console. This page has two tabs (**ASP Server** and **Web Server**), which you use to access settings for the ASP Server and the Web server.

Note

The Administration Console is not available with Sun Chili!Soft ASP for Windows. On Windows NT and Windows 2000, regedit32 can be used to edit some of the Sun Chili!Soft ASP configuration settings. For more information, see "Editing the Windows Registry" in this chapter.

On Windows NT and Windows 2000, Sun Chili!Soft ASP runs automatically whenever an ASP page is requested by a user (provided the Web server is running). Sun Chili!Soft runs until the Web server is stopped. When the Web server is restarted, Sun Chili!Soft ASP will not run until an ASP page is requested by a user.

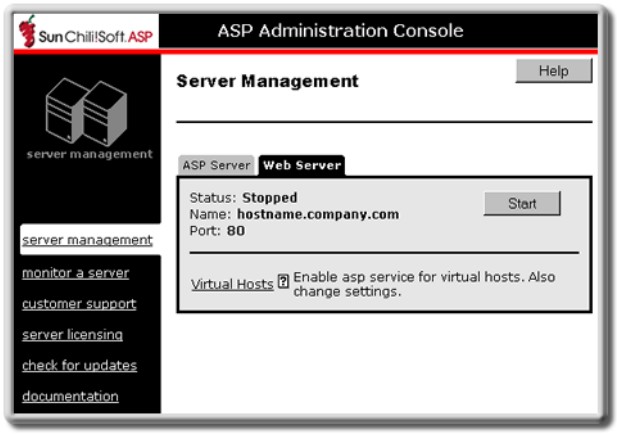
The **ASP Server** tab of the **Server Management** page displays when you open the Administration Console.



The **ASP Server** tab displays the following items:

- **Status** indicates whether the ASP Server is running or stopped.
- **Uptime** indicates the length of time since the ASP Server was started or restarted.
- **Location** indicates the directory in which the ASP Server is installed.
- **Stop**, **Start**, and **Restart** buttons enable you to stop, start, and restart the ASP Server. For more information, see "Stopping and Restarting the ASP Server" in this chapter.
- The **ASP Applications** link displays settings for adding, removing, and configuring ASP applications. For more information, see "Configuring ASP Applications" in this chapter.
- The **Components** link takes you to the page where you can enable or disable SpicePack and Chili!Beans components. For more information, see "Enabling External Components" in this chapter.
- The **Databases** link takes you to database configuration settings. For more information, see "Configuring a Database" in this chapter.
- The **Settings** link takes you to general ASP Server settings. For more information, see "Changing ASP Server Settings" in this chapter.
- The **View Logs** link takes you to pages where you can view the log files enabled for the ASP Server. For more information, see "Viewing Log Files" in this chapter.

When you click the **Web Server** tab, the following page displays:



The **Web Server** tab displays the following items:

- **Status** indicates whether the Web server is running or stopped.
- **Name** is the Web server hostname.
- **Port** is the port the Web server is using.
- **Stop**, **Start**, and **Restart** buttons enable you to stop, start, and restart the Web server. For more information, see "Starting and Stopping the Web Server" in this chapter.
- The **Virtual Hosts** link takes you to an option for enabling and disabling ASP processing for individual virtual hosts. For more information, see "Enabling ASP for a Virtual Host" in this chapter.

Note

On Cobalt systems, the Web server is managed from the Cobalt Administration Console.

Changing ASP Server Settings

The Sun Chili!Soft ASP Administration Console **Server Settings** page provides access to the basic configuration settings for the ASP Server. You can change these settings using the following procedure.

For changes to take effect, you must restart the ASP Server. Restarting the ASP Server resets all **Session** and **Application** variables.

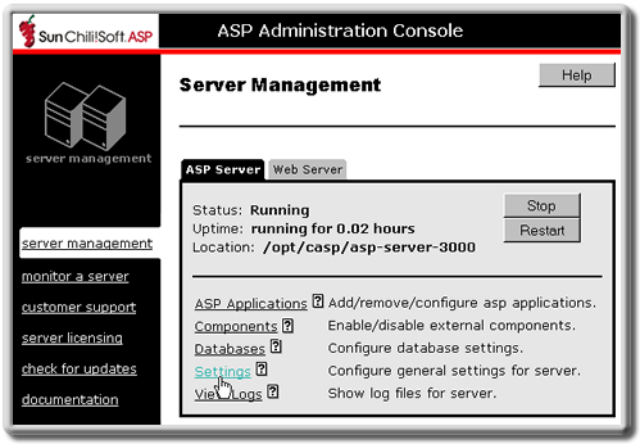
To change ASP Server settings

1. Open the Administration Console by using the following URL:

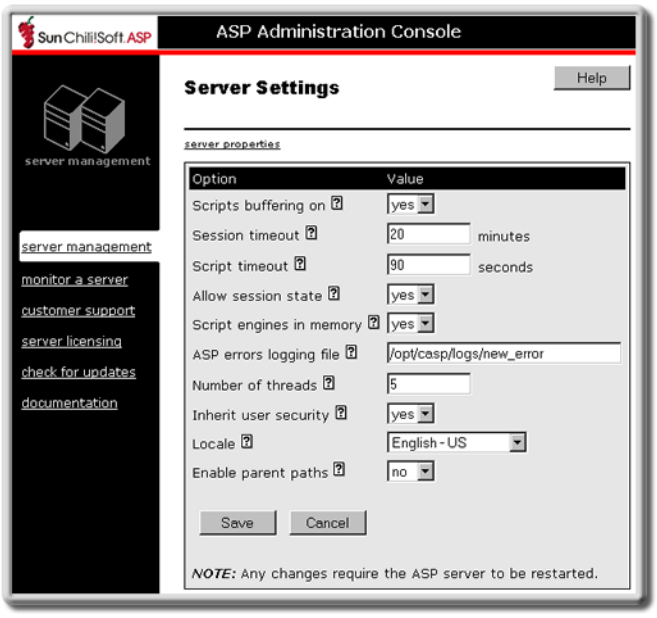
   http://[HOSTNAME]:[PORT]

   where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

2. On the **ASP Server** tab of the **Server Management** page (the first page to display when you open the console), click **Settings**.



The **Server Settings** page displays.



3. Configure the settings as desired (settings are described below).
4. When finished, click **Save** to save your changes.

   – or –

   Click **Cancel** to revert to the last settings that were saved.

   The **Server Management** page displays.
5. To put your changes into effect, restart the ASP Server by clicking **Restart**.

Note

Restarting the ASP Server resets all **Session** and **Application** variables.

ASP Server Setting	Explanation
Scripts buffering on	**Yes** enables scripts buffering. The ASP Server processes an entire ASP page before returning its HTML output to the browser, yielding better server performance. **No** disables scripts buffering. The ASP Server returns the HTML output for an ASP page to the browser incrementally, as soon as the HTML is processed, making debugging easier. This setting is **yes** by default. For more information, see "Enabling Scripts Buffering" in this chapter.
Session timeout	This specifies the number of minutes the ASP Server maintains a user's session information since the last page request. When a user does not submit a page request for the specified length of time, the server cancels the session and discards its information. If a value for **SessionTimeout** is specified in the script, it overrides this setting. This setting is **20** minutes by default. For more information, see "Changing the Session Timeout Value" in this chapter.
Script timeout	This specifies the number of seconds the ASP Server waits for a page to finish processing before it cancels the page request. A value for **ScriptTimeout** specified in a script is used only if it is higher than this value. This setting is **90** seconds by default. For more information, see "Changing the Script Timeout Value" in this chapter.
Allow session state	This specifies whether the ASP Server maintains session state. This setting must be enabled (**yes**) for **Session** objects in scripts to function. This setting is **yes** by default. For more information, see "Enabling Session State" in this chapter.
Script engines in memory	**Yes** enables ASP scripts to be cached in memory, so ASP pages are served faster. **No** disables caching, which reduces the system memory used by the server. This setting is **yes** by default. For more information, see "Enabling Script Caching" in this chapter.
ASP errors logging file	To enable logging for the ASP Server and specify the location of the log file, type the absolute path name of the log file in this text box. Sun Chili!Soft ASP creates the log file in the directory you specify. You cannot give the log file the same name as a file that already exists in that directory. If the **ASP errors logging file** text box is empty (the default), no logging is performed. For more

information, see "Enabling ASP Error Logging" in this chapter.

Number of threads

This specifies the number of simultaneous threads the ASP Server handles. The number of threads is **10** by default. If you have many ASP pages that include blocking operations (database access, for example), it is best to increase this number. However, keep in mind that increasing the number of threads also increases system overhead. For more information, see "Configuring Multi-threading" in this chapter.

Inherit user security

When **Inherit user security** is set to **yes**, the ASP Server runs with the permissions of the Apache Web server or the virtual host defined in the Apache Web server's httpd.conf file. This is the default security mode for Sun Chili!Soft ASP. However, if you are running iPlanet Web Server or Zeus Web Server, be sure to read the security note that follows.

When **Inherit user security** is set to **no**, the ASP Server runs with the permissions of the user who started the ASP Server, unless a different user or group is specified in the Sun Chili!Soft ASP configuration file, casp.cnfg. This can create a security risk for your server. If you change **Inherit user security** to **no**, be sure to specify a user or group in casp.cnfg, as described in "Editing the Chili!Soft Configuration File" in "Chapter 3: Managing Sun Chili!Soft ASP."

Important Security Information about iPlanet and Zeus Web Servers

iPlanet Web Server and Zeus Web Server do not support Inherit User Security mode, even when **Inherit user security** is set to **yes** in the Administration Console.

To protect the security of your server, when running Sun Chili!Soft ASP with these Web servers, you should specify a user or group in the casp.cnfg file, as described in "Editing the Chili!Soft Configuration File" in "Chapter 3: Managing Sun Chili!Soft ASP." The ASP Server then runs with the permissions of that user or group.

For more information, see "Securing the Server" in this chapter.

Note: ADO logging will not be functional if **Inherit user security** is set to **yes**.

Locale	This specifies the locale setting. The ASP Server uses the appropriate code page for the language associated with the locale specified here. It also correctly formats dates, numbers, and currency according to the locale. For more information, see "Configuring International Support" in this chapter. (Supported locales vary by platform.)
Enable parent paths	This enables file system access by an ASP application to a directory in the file system that is not contained in the ASP application root directory or its subdirectories. By default, **Enable parent paths** is set to **no**. This is the most secure setting and is appropriate for most shared Web hosting environments. Changing **Enable parent paths** to **yes** can affect the security of your server. For more information, see "Configuring File System Access" in this chapter.

Stopping and Restarting the ASP Server

There are times when you need to stop and restart the ASP Server (for example, you need to stop the ASP Server to perform a product upgrade). You must restart the ASP Server to put ASP Server configuration settings into effect. To stop or restart the ASP Server, use the following procedure.

Note

Restarting the ASP Server resets all **Session** and **Application** variables.

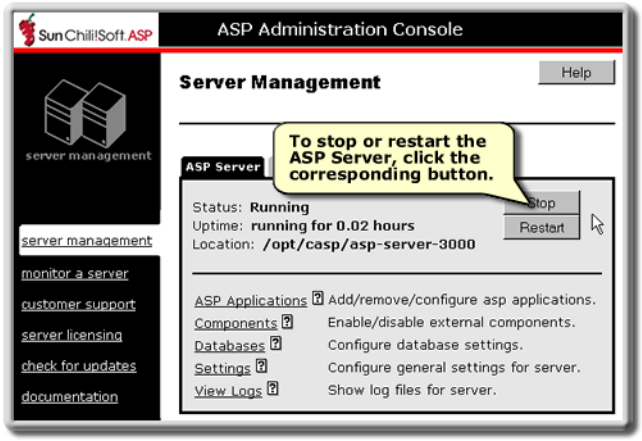
To stop, start, or restart the ASP Server

1. Open the Administration Console by using the following URL:

   http://[HOSTNAME]:[PORT]

   where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

2. On the **Server Management** page (the first page to display when you open the Administration Console), click **Stop**, **Start**, or **Restart** as desired.



Stopping, starting, or restarting can take from several seconds to about one minute to execute.

Configuring International Support

You might want to use the Sun Chili!Soft ASP Server to serve Web pages in languages other than United States (US) English or in countries other than the US. If so, you can change the locale setting. When you do this, the ASP Server uses the appropriate code pages for the language associated with the locale you specify. It also correctly formats dates, numbers, and currency according to the locale. Depending on your operating system, you can specify locales for the following languages:

- English - US
- English - British
- Dutch
- French
- German
- Japanese Shift-JIS
- Spanish
- Swedish

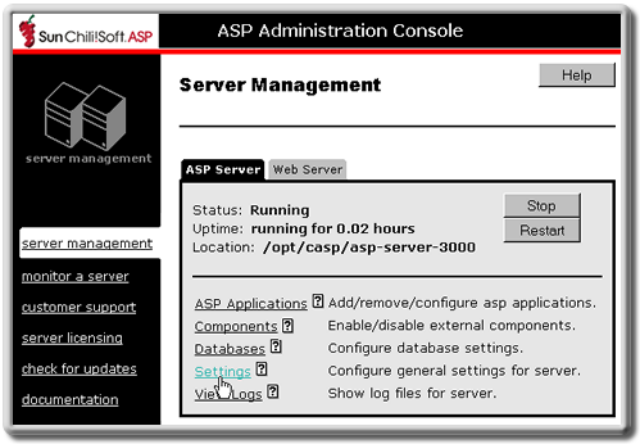
To change the Sun Chili!Soft ASP locale setting

1. Open the Administration Console by using the following URL:

   http://[HOSTNAME]:[PORT]

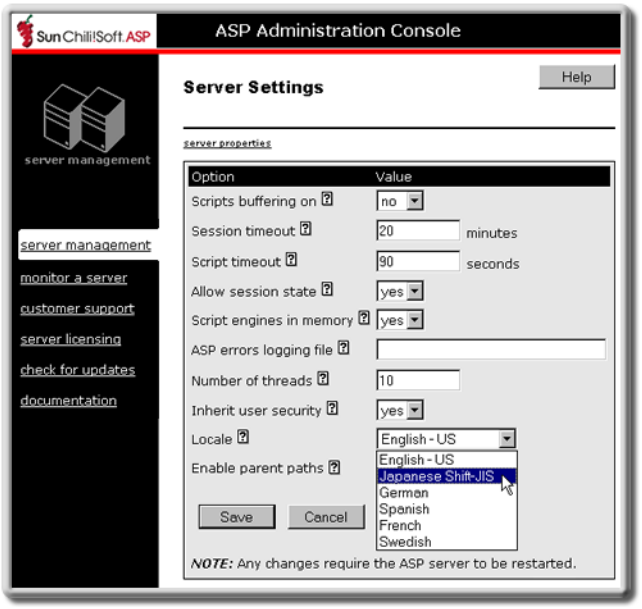
   where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

2. On the **ASP Server** tab of the **Server Management** page (the first page to display when you open the Administration Console), click **Settings**.



The **Server Settings** page displays.

3. In the **Locale** drop-down list, select the desired locale.



4. Click **Save** to save your changes.

   – or –

   Click **Cancel** to revert to the last settings that were saved.

   The **Server Management** page displays.

5. To put your changes into effect, restart the ASP Server by clicking **Restart**.

Note

Restarting the ASP Server resets all **Session** and **Application** variables.

See also:

Developing International Applications in "Chapter 4: Building a Sun Chili!Soft ASP Application"

Enabling Session State

With Sun Chili!Soft ASP you can specify whether the ASP Server maintains session state. Session state is enabled by default. To conserve system resources, you might want to disable this feature. However, for the **Session** object used in scripts to function, session state must be enabled.

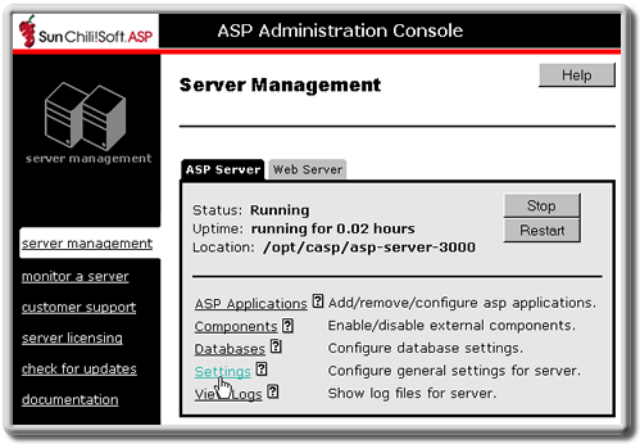
To enable or disable session state

1. Open the Administration Console by using the following URL:

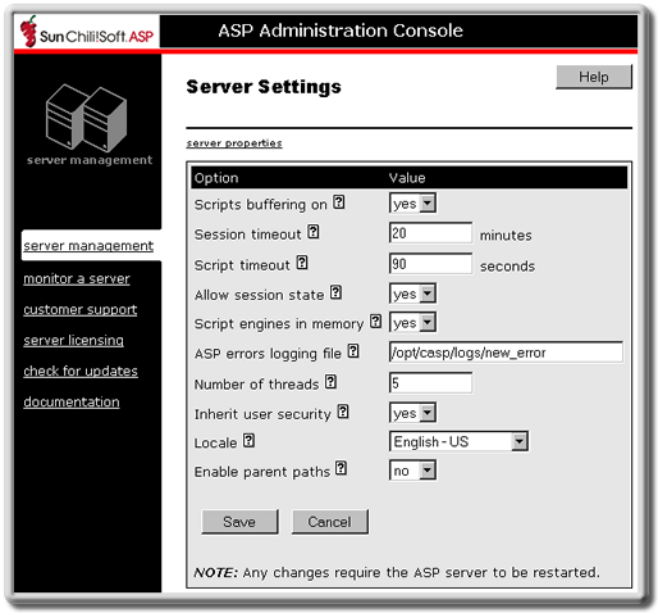
   http://[HOSTNAME]:[PORT]

   where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

2. On the **ASP Server** tab of the **Server Management** page (the first page to display when you open the Administration Console), click **Settings**.



   The **Server Settings** page displays.



3. In the **Allow session state** drop-down list, select **yes** or **no** as desired.
4. Click **Save** to save your changes.

   – or –

   Click **Cancel** to revert to the last settings that were saved.

   The **Server Management** page displays.
5. To put your changes into effect, restart the ASP Server by clicking **Restart**.

Note

Restarting the ASP Server resets all **Session** and **Application** variables.

See also:

ASP Session Object Overview in "Chapter 4: Building a Sun Chili!Soft ASP Application"

ASP Session Object in "Chapter 5: Developer's Reference"

Enabling External Components

The Sun Chili!Soft ASP Administration Console **Components** page provides access to external SpicePack and Chili!Beans components.

Sun Chili!Soft SpicePack is a set of Component Object Model (COM) components that handle commonly used ASP application functionality. The components are Chili!Mail, Chili!POP3, and Chili!Upload.

The Chili!Beans ActiveX control is a wrapper that enables Java objects to be used by COM controllers, such as ActiveX scripting engines like VBScript.

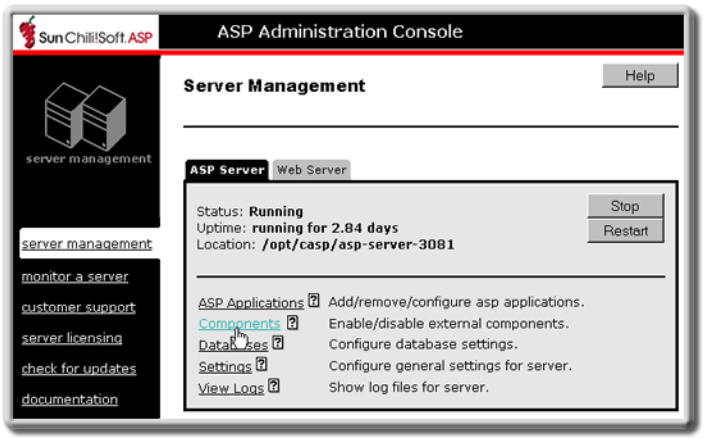
To enable external components

1. Open the Administration Console by using the following URL:

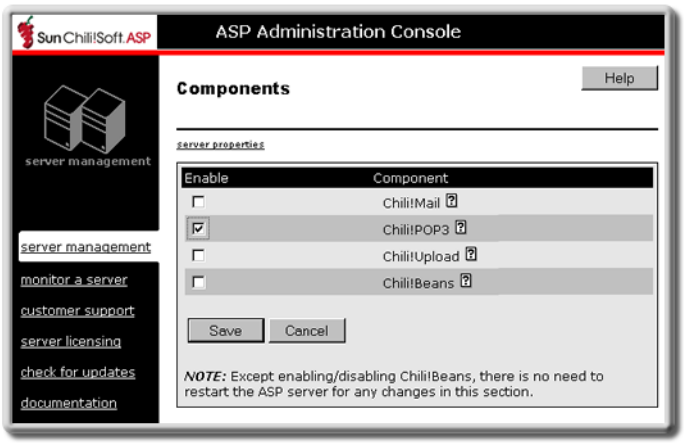
   http://[HOSTNAME]:[PORT]

   where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

2. On the **ASP Server** tab of the **Server Management** page (the first page to display when you open the console), click **Components**.



   The **Components** page displays.



3. Click to select or clear (enable or disable) the components as desired.

   Note: If you did not enable Chili!Beans support during installation (install a Java runtime environment), **Chili!Beans** will not be listed on the **Components** page. See "Enabling Java Support" in "Chapter 2: Installing and Configuring Sun Chili!Soft ASP."

4. When finished, click **Save** to save your changes.

   – or –

   Click **Cancel** to revert to the last settings that were saved.

   The **Server Management** page displays.

5. If you changed the status of the Chili!Beans component, you must restart the ASP Server by clicking **Restart** on the **Server Management** page. You do not need to restart the ASP Server if you changed the status of the other components.

Note

Restarting the ASP Server resets all **Session** and **Application** variables.

See also:

SpicePack Component Reference in "Chapter 5: Developer's Reference"

Chili!Beans Component Reference in "Chapter 5: Developer's Reference"

Securing the Server

The following topics address security issues for the Sun Chili!Soft ASP Server:

- Configuring File System Access
- Setting the Security Mode
- Disabling Performance Monitoring

Configuring File System Access

You might want to enable access by an ASP application to a directory in the file system that is not contained in the ASP application root directory or its subdirectories. This type of access is configured from the Sun Chili!Soft ASP Administration Console using the **Enable parent paths** setting.

By default, **Enable parent paths** is set to **no**. When **Enable parent paths** is set to **no**, a **FileSystemObject** instantiated by an ASP application is limited to that application's defined directory. In this case, **#include** statements cannot use the **"..\"** syntax to access files outside the ASP application root directory. This is the most secure setting, and is appropriate for most shared Web hosting environments. (Unlike Sun Chili!Soft ASP, with Microsoft ASP, when **Enable parent paths** is set to **no**, you can still create a text file outside of the application directory.)

When **Enable parent paths** is set to **yes**, **FileSystemObject** can access files outside the ASP application directory. In this scenario, ASP developers can use the **"..\"** syntax in **#include** statements to access any file outside of the Web directory that the ASP Server has file system permission to read.

Warning! Important Security Information

Changing **Enable parent paths** to **yes** can affect the security of your server. Before you change this setting, make sure that your ASP Server has permission to access only the files you want to be publicly accessible, and that it does not have access to sensitive files containing configuration or password information. You can restrict the permissions of the ASP Server by defining the user

it runs under, and making sure that that user has appropriately restricted file system permissions. For more information, see "Setting the Security Mode."

To configure file system access

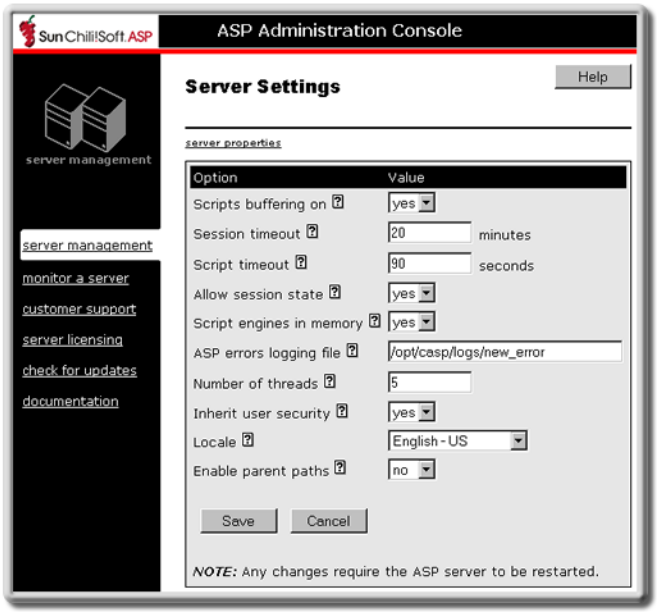
1. Open the Administration Console by using the following URL:

   http://[HOSTNAME]:[PORT]

   where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

2. On the **ASP Server** tab of the **Server Management** page (the first page to display when you open the console), click **Settings**.

   The **Server Settings** page displays.



3. In the **Enable parent paths** drop-down list, select **yes** or **no**.

See also:

Defining ASP Applications on the Server in "Chapter 2: Installing and Configuring Sun Chili!Soft ASP"

Using Server-side Includes in "Chapter 4: Building a Sun Chili!Soft ASP Application"

Setting the Security Mode

With Sun Chili!Soft ASP for Linux, Cobalt, and UNIX-based systems, you can configure the Sun Chili!Soft ASP Server to run under either Defined User Security mode or Inherit User Security mode. The appropriate mode depends on your Web hosting environment, and has important security implications for your server. Be sure to read "Important Security Information" later in this topic, particularly if you are running a Zeus or iPlanet Web server.

- **Inherit User Security** mode is available only for Sun Chili!Soft ASP running with Apache. This mode is useful in shared Web hosting environments because the ASP Server runs with the permissions of the user defined for the Apache Web server. In a Web hosting environment using virtual hosts, the ASP Server runs as the user configured for the virtual host. For example, if the Web server is configured to run as user "john," when someone accesses the virtual server www.johns-site.com, the ASP Server runs under the account "john" when processing ASP page requests for www.johns-site.com. You can enable this mode from the Sun Chili!Soft ASP Administration Console, as described later in this topic

  Note that ADO logging will not be functional if **Inherit user security** is set to **yes**. For information about ADO logging, see "Enabling ADO Logging" in this chapter.

- **Defined User Security** mode is appropriate for most corporate or dedicated Web hosting environments. In this mode, the ASP Server runs with the permissions of the user and group defined in the Sun Chili!Soft ASP configuration file, casp.cnfg. The user and group account under which the ASP Server is configured to run should have access rights to all *.asp and *.asa pages, and should also have rights to Sun Chili!Soft ASP configuration files, such as casp.cnfg and ODBC.INI. You enable this mode by setting **Inherit user security** to **no** in the Sun Chili!Soft ASP Administration Console, and then specifying a user and group in the casp.cnfg file, as described later in this topic.

  Note that even if a user or group is specified in casp.cnfg, if **Inherit user security** is set to **yes** in the Administration Console, the ASP Server runs under Inherit User Security mode.

Important Security Information

If you set **Inherit user security** to **no** and do not specify a user and group in the casp.cnfg file, the ASP Server runs as root. This can compromise the security of your server.

IPlanet and Zeus Web servers do not support Inherit User Security mode, even when **Inherit user security** is set to **yes** in the Administration Console. To protect the security of your server when running Sun Chili!Soft ASP with these Web servers, you should specify a user or group in the casp.cnfg file, as described in "Editing the Chili!Soft Configuration File" in "Chapter 3: Managing Sun Chili!Soft ASP." The ASP Server then runs with the permissions of that user or group.

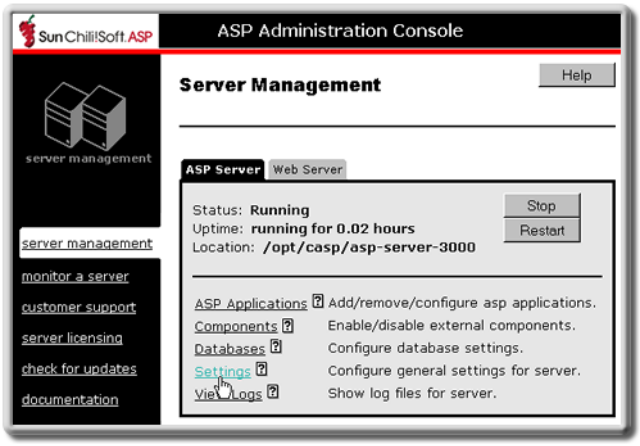
To set the ASP Server security mode

1. Open the Administration Console by using the following URL:

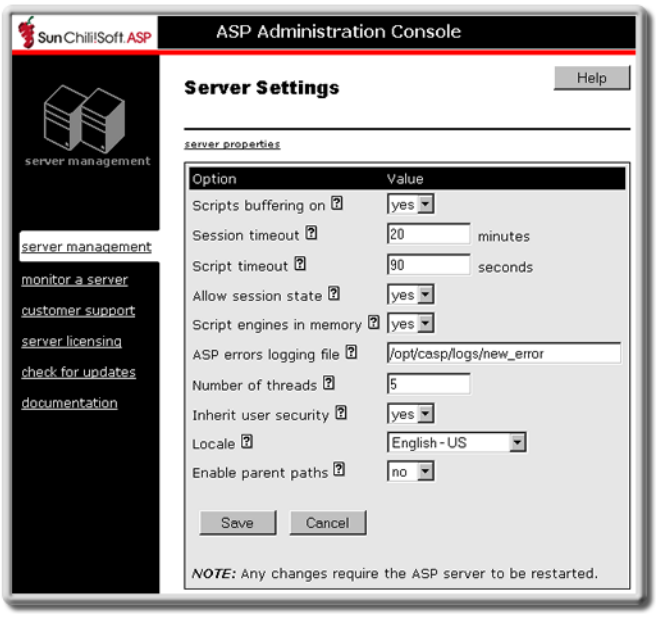
   http://[HOSTNAME]:[PORT]

   where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

2. On the **ASP Server** tab of the **Server Management** page (the first page to display when you open the Administration Console), click **Settings**.



The **Server Settings** page displays.



3. In the **Inherit user security** drop-down list, select **yes** to run under Inherit User Security mode, or **no** to run under Defined User Security mode. If you select **no**, you should edit the casp.cnfg file to add a user or group for the ASP Server to run under, as described in "Editing the Sun Chili!Soft ASP Configuration File" in this chapter. If you do not make that change, the ASP Server runs as root, which can compromise the security of your server. You should always run Web servers other than Apache under Defined User Security Mode.
4. Click **Save** to save your changes.

   – or –

   Click **Cancel** to revert to the last settings that were saved.

   The **Server Management** page displays.
5. To put your changes into effect, restart the ASP Server by clicking **Restart**.

Note

Restarting the ASP Server resets all **Session** and **Application** variables.

See also:

Configuring File System Access in this chapter

Disabling Performance Monitoring

If you are running Sun Chili!Soft ASP for UNIX or Linux in a shared Web hosting environment, it is strongly recommended that you disable server performance monitoring to protect the security of your server. Disabling performance monitoring is described in this topic.

By default, Sun Chili!Soft ASP monitors server performance and displays this information on the Sun Chili!Soft ASP Administration Console **Server Monitoring** page, as described in "Monitoring the ASP Server" in this chapter.

Note

This feature is not available on Windows systems.

Sun Chili!Soft ASP stores the server performance information in the following files:

/tmp/.casp[PORT]/chili-psm

/tmp/.casp[PORT]/.pm-chili-psm

/tmp/.pm-chili-psm

/tmp/chili-psm

These files are created with "world-readable" permissions that might not be appropriate in a shared Web hosting environment. You can disable performance monitoring and the creation of these log files by editing the **enablemonitoring** setting in the `[default machine]` section of the Sun Chili!Soft ASP configuration file, casp.cnfg. When you do this, server performance information is no longer displayed on the **Server Monitoring** page of the Administration Console.

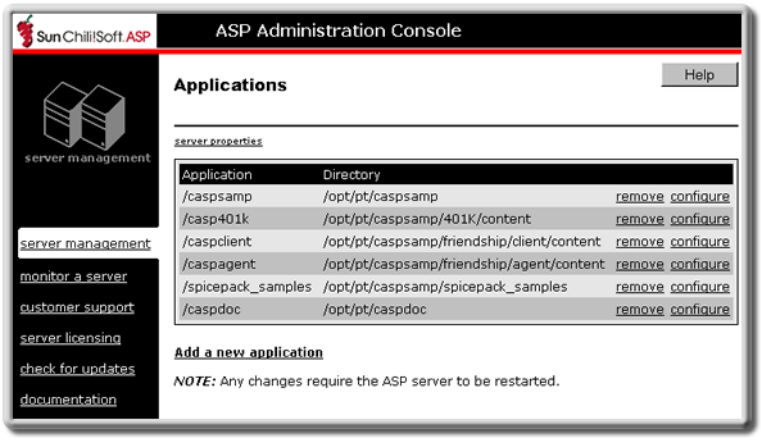
For more information about editing casp.cnfg, see "Editing the Sun Chili!Soft ASP Configuration File" in this chapter.

Configuring ASP Applications

As discussed in "Defining ASP Applications on the Server" in "Chapter 2: Installing and Configuring Sun Chili!Soft ASP," you must define an ASP application on the ASP Server for the application to be recognized and processed when a user requests an ASP page. The easiest way to define and configure an application is by using the Sun Chili!Soft ASP Administration Console, as discussed in this section. However, if you need to configure an application in a hosted environment, see the instructions in "Running Applications in a Shared Web Hosting

Environment" in this chapter. For more information about defining Microsoft FrontPage applications, see "Using the FrontPage Services File in a Shared Environment" in this chapter.

You can define and configure an ASP application from the Administration Console **Applications** page.



- **Add a new application** creates a new application and associates it with the physical directory containing the global.asa file. See "Adding an ASP Application" in this chapter.
- **remove** removes an ASP application from the ASP Server. See "Removing an ASP Application" in this chapter.
- **configure** associates an ASP application with a physical directory containing the global.asa file. See "Editing ASP Application Settings" in this chapter.

Note

On Windows NT and Windows 2000, ASP applications are defined by adding aliases or virtual directories to the Web server. Sun Chili!Soft ASP treats each alias or virtual directory as an ASP application. With Apache Web Server, ASP applications are defined by adding an alias to the httpd.conf file. With iPlanet Web Server, ASP applications are defined by adding an "additional document directory" using the server's Administration tool.

Adding an ASP Application

For the ASP Server to process an ASP application when a user requests an ASP page, the ASP application must be defined on the ASP Server. The easiest way to do this is by using the Sun Chili!Soft ASP Administration Console. From the console, you "add" an application by giving the application a name and specifying the physical directory containing the application files and the global.asa file. When you do this, a virtual directory for the application is created on the Web server and associated with the physical directory containing the application files.

To define an application for a virtual host, instead of using the following procedure, use the instructions in "Enabling ASP for a Virtual Host" in this chapter. If the Web developers you

support use Microsoft FrontPage, see the description of FrontPage applications in "Using the FrontPage Services File in a Shared Environment" in this chapter.

Note

On Windows NT and Windows 2000, ASP applications are defined by adding aliases or virtual directories to the Web server. Sun Chili!Soft ASP treats each alias or virtual directory as an ASP application. With the Apache Web Server, ASP applications are defined by adding an alias to the httpd.conf file. With iPlanet Web Server, ASP applications are defined by adding an "additional document directory" using the Web server's Administration tool.

To define an ASP application that will not be served by a virtual host, use the following procedure.

To add an ASP application to the ASP Server

1. Open the Administration Console by using the following URL:

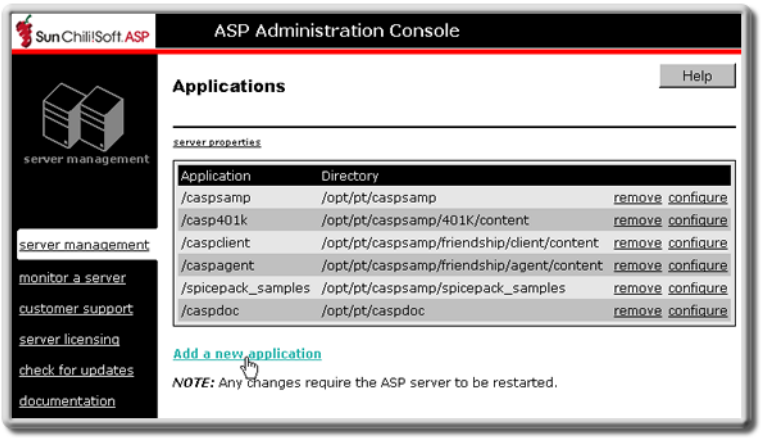
   http://[HOSTNAME]:[PORT]

   where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

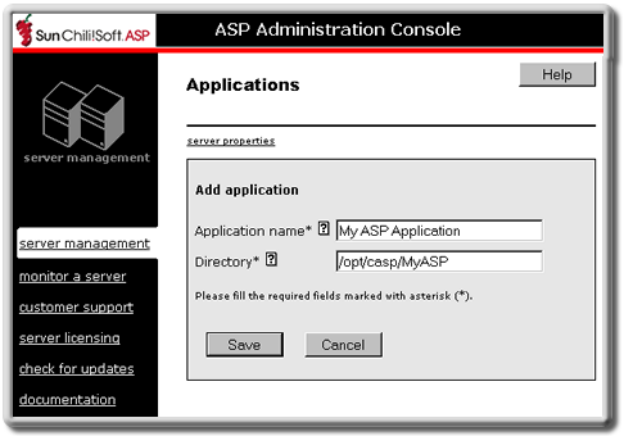
2. On the **ASP Server** tab of the **Server Management** page (the first page to display when you open the Administration Console), click **ASP Applications**.

   The **Applications** page displays, showing a list of currently configured ASP applications.

3. Click **Add a new application**.



   The **Add application** page displays.



4. In the **Application name** box, type the name of the virtual directory to create and enable as an ASP application.

   Note: On Cobalt RaQ4, the application name must be the same as the name of the physical directory containing the application files. You do not have the option to specify a directory. When you enter the application name, Sun Chili!Soft ASP automatically creates a directory on your hard disk with the same name, if it does not already exist. You can then copy your application files into this directory.

5. In the **Directory** box, type the absolute path name of the application directory. The application directory is the top-level physical directory that contains the application ASP files, the global.asa file (if one is being used for this application), and any application subdirectories. Note that this step does not apply to Cobalt RaQ4 (see the note in the previous step).

6. Click **Save**.

   – or –

   Click **Cancel** to cancel any entries.

   The **Applications** page displays.

7. In the left navigation pane, click **server management**.

   The **Server Management** page displays.

8. To put your changes into effect, restart the ASP Server by clicking **Restart**.

Note

Restarting the ASP Server resets all **Session** and **Application** variables.

See also:

Defining ASP Applications on the Server in "Chapter 2: Installing and Configuring Sun Chili!Soft ASP"

Configuring ASP Applications in this chapter

Editing ASP Application Settings in this chapter

Removing an ASP Application in this chapter

Removing an ASP Application

If you want the ASP Server to stop processing an ASP application, you must remove the ASP application from the ASP Server, as described in the following procedure. This deletes the virtual directory for the application from the Web server. It does not delete the physical directory containing the application files. For more information about ASP applications, see "Configuring ASP Applications" in this chapter.

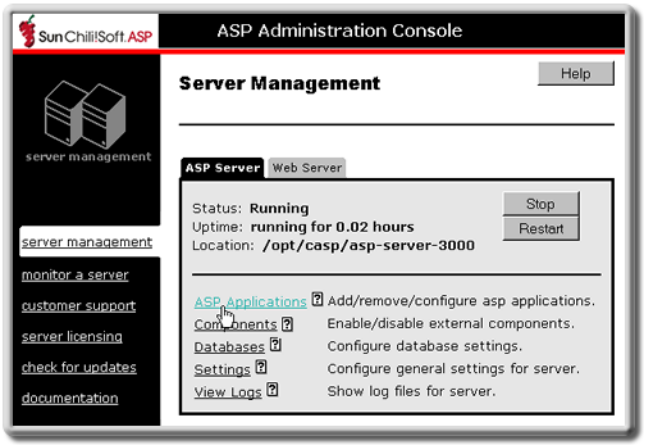
To remove an ASP application from the ASP Server

1. Open the Administration Console by using the following URL:

   http://[HOSTNAME]:[PORT]

   where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

2. On the **ASP Server** tab of the **Server Management** page (the first page to display when you open the Administration Console), click **ASP Applications**.



   The **Applications** page displays, showing a list of currently configured ASP applications.

3. In line with the application that you want to remove, click **remove**.

4. When prompted to confirm removing the application, click **Yes**.

   The **Applications** page displays.

5. In the left navigation pane, click **server management**.

6. To put your changes into effect, restart the ASP Server by clicking **Restart**.

Note

Restarting the ASP Server resets all **Session** and **Application** variables.

See also:

Defining ASP Applications on the Server in "Chapter 2: Installing and Configuring Sun Chili!Soft ASP"

Configuring ASP Applications in this chapter

Editing ASP Application Settings in this chapter

Adding an ASP Application in this chapter

Editing ASP Application Settings

For the ASP Server to process an ASP application when a user requests an ASP page, you must first add it to the ASP Server, as described in "Adding an ASP Application" in this chapter. Later, if you want to change the application name (for example, the virtual directory name) or the physical directory associated with the application, you can use the following procedure to do so. For more information about ASP applications, see "Configuring ASP Applications" in this chapter.

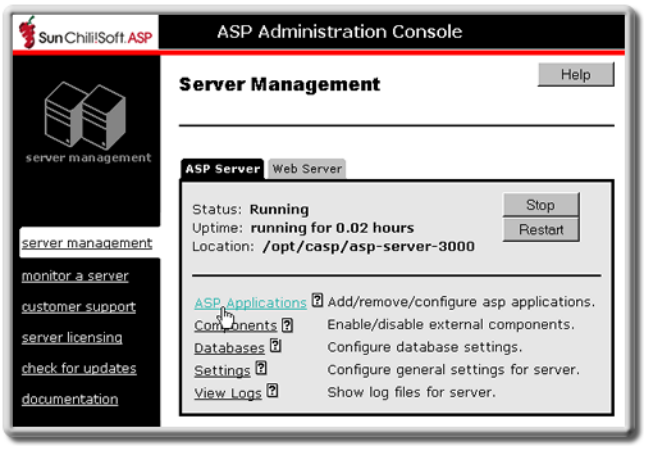
To edit ASP application settings

1. Open the Administration Console by using the following URL:

   http://[HOSTNAME]:[PORT]

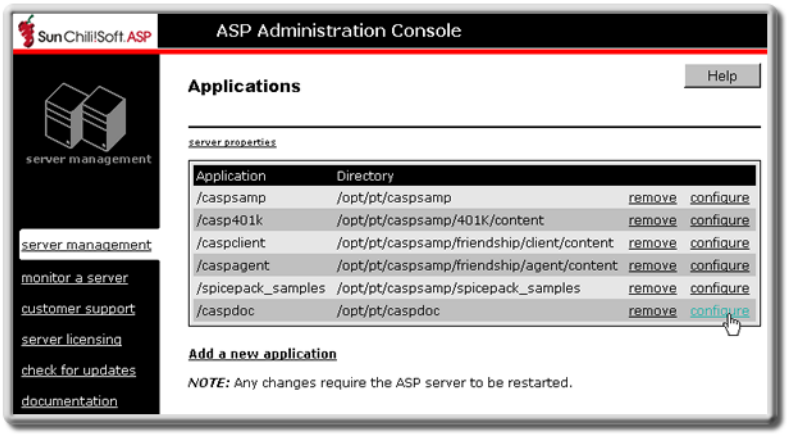
   where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

2. On the **ASP Server** tab of the **Server Management** page (the first page to display when you open the Administration Console), click **ASP Applications**.

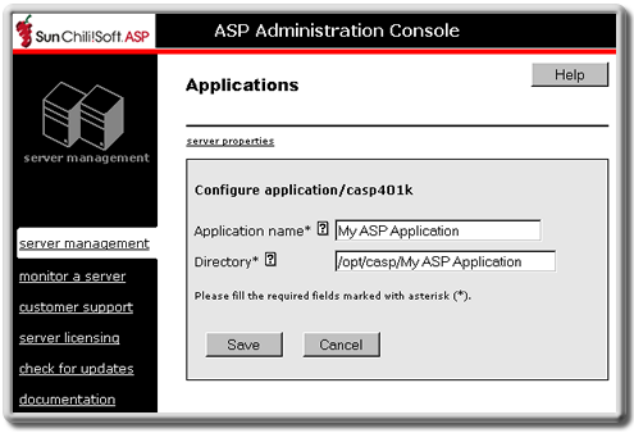


   The **Applications** page displays, showing a list of currently configured ASP applications.

3. In line with the application that you want to edit, click **configure**.



The **Configure application** page displays.



4. If you want to change the application name, type the new name in the **Application name** box.
5. If you want to change the physical directory associated with the application, type the absolute path name of the new directory in the **Directory** box. The application directory is the top-level directory that contains the application files, optional global.asa file, and any application subdirectories.
6. Click **Save**.

   – or –

   Click **Cancel** to cancel any changes.

   The **Applications** page displays.
7. In the left navigation pane, click **server management**.
8. To put your changes into effect, restart the ASP Server by clicking **Restart**.

Note

Restarting the ASP Server resets all **Session** and **Application** variables.

See also:

Defining ASP Applications on the Server in "Chapter 2: Installing and Configuring Sun Chili!Soft ASP"

Configuring ASP Applications in this chapter

Adding an ASP Application in this chapter

Removing an ASP Application in this chapter

Viewing Information about the ASP Server

Sun Chili!Soft ASP provides several options for viewing information about the ASP Server. It enables you to monitor real-time performance data, view diagnostic information, and log ASP errors.

In this section:

- Monitoring ASP Server Performance
- Enabling ASP Error Logging
- Viewing the ASP Errors Log
- Viewing Server Diagnostics

See also:

Optimizing Server Performance in this chapter

Monitoring ASP Server Performance

On UNIX and Linux systems, the **Server Monitoring** page of the Sun Chili!Soft ASP Administration Console displays real-time information about ASP Server performance.

Note

This feature is not available with Sun Chili!Soft ASP for Windows. On Windows systems, performance monitoring information is available via the Windows NT or Windows 2000 Performance Monitor. See your Microsoft documentation for more information.

If you have disabled performance monitoring, as described in "Disabling Performance Monitoring" in this chapter, you cannot view this server performance information. Disabling server performance monitoring is a recommended security precaution if you are running Sun Chili!Soft ASP in a shared Web hosting environment.

To view real-time information about the ASP Server

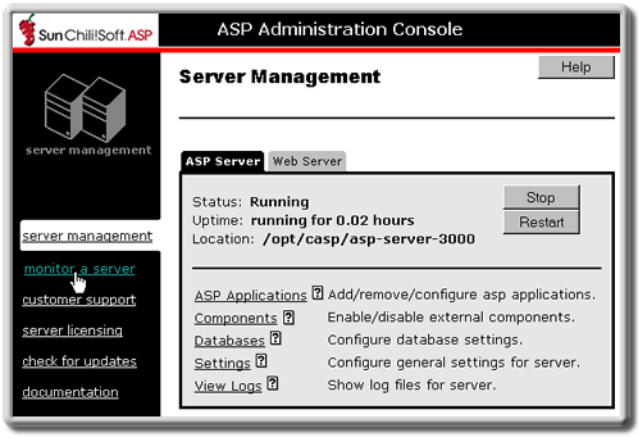
1. Open the Administration Console by using the following URL:

   http://[HOSTNAME]:[PORT]

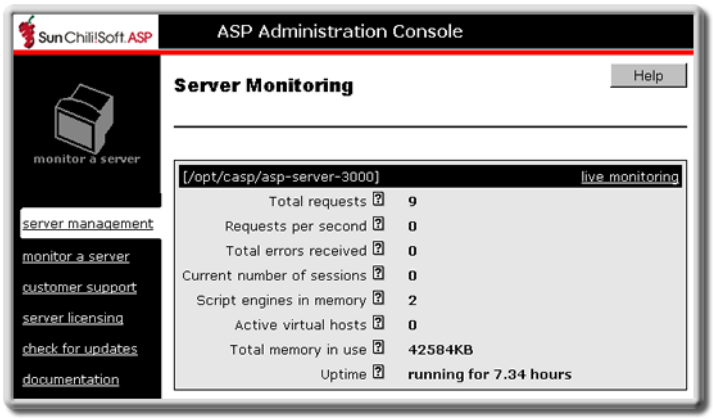
where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

The **Server Management** page displays.

2. In the left navigation pane, click **monitor a server**.



The **Server Monitoring** page displays.



3. To continuously monitor the server, click **live monitoring**. This opens a separate window that displays constantly updated information.

The **Server Monitoring** page displays the following information:

Item	Explanation
Total requests	Total number of requests since the ASP Server was started
Requests per second	Number of requests per second being processed by the ASP Server
Total errors received	Number of ASP Server errors logged since the server was started
Current number of sessions	Number of sessions currently active on the ASP Server
Script engines in memory	Number of ASP scripts currently cached by the ASP Server

Active virtual hosts	Number of virtual hosts that currently have one or more active sessions
Total memory in use	System memory (RAM) currently being used by the ASP Server
Uptime	Length of time the ASP Server has been running since the last restart

See also:

Changing ASP Server Settings in this chapter

Viewing Server Diagnostics in this chapter

Enabling ASP Errors Logging in this chapter

Viewing the ASP Errors Log in this chapter

Optimizing Server Performance in this chapter

Enabling ASP Errors Logging

For Sun Chili!Soft ASP to log ASP errors, you must first enable logging. For more information about viewing the log file, see "Viewing the ASP Errors Log" in this chapter.

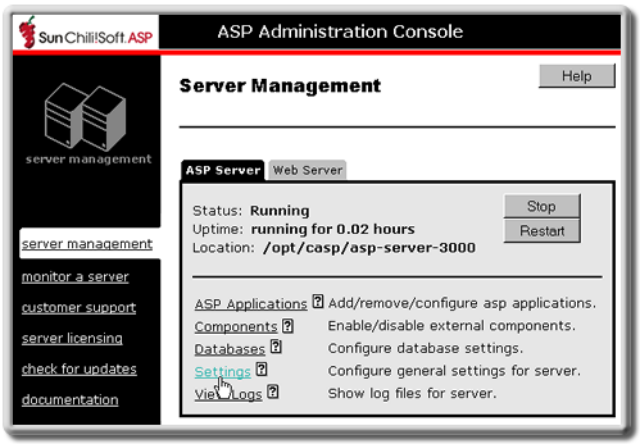
To enable ASP errors logging

1. Open the Administration Console by using the following URL:

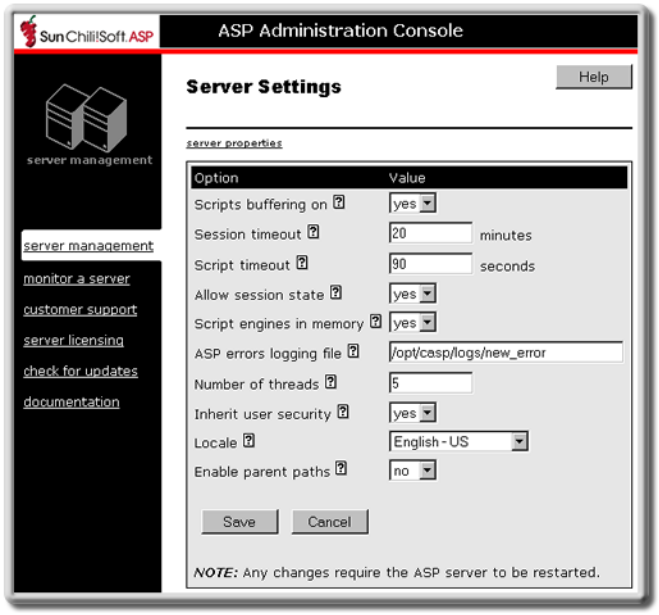
   http://[HOSTNAME]:[PORT]

   where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

2. On the **ASP Server** tab of the **Server Management** page (the first page to display when you open the Administration Console), click **Settings**.



The **Server Settings** page displays.



3. In the **ASP errors logging file** box, type the name of the log file to which you want ASP errors logged. You cannot give the log file the same name as a file that already exists in the directory. If the **ASP errors logging file** box is empty (the default), no logging is performed.
4. Click **Save** to save your changes.

   – or –

   Click **Cancel** to revert to the last settings that were saved.

   The **Server Management** page displays.
5. To put your changes into effect, restart the ASP Server by clicking **Restart**.

Note

Restarting the ASP Server resets all **Session** and **Application** variables.

A log file with the name you specified is created in the following directory:

/[C-ASP_INSTALL_DIR]/logs

where [C-ASP_INSTALL_DIR] is the path name of the Sun Chili!Soft ASP installation directory (/opt/casp by default).

See also:

Monitoring ASP Server Performance in this chapter

Optimizing Server Performance in this chapter

Viewing Information About the ASP Server in this chapter

Viewing the ASP Errors Log

You can view the ASP errors log from the **ASP Server** tab of the **Server Management** page of the Sun Chili!Soft ASP Administration Console. To log ASP errors and view the logging information, you must first enable logging as described in "Enabling ASP Errors Logging" in this chapter.

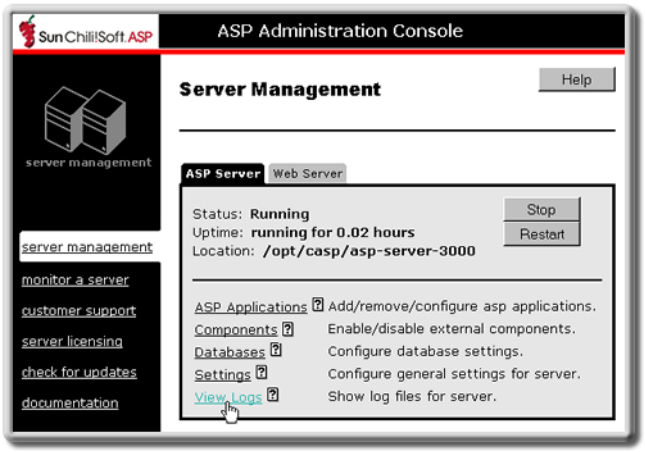
To view the ASP errors log

1. Open the Administration Console by using the following URL:

   http://[HOSTNAME]:[PORT]

   where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

2. On the **ASP Server** tab of the **Server Management** page (the first page to display when you open the Administration Console), click **View Logs**.



   The **Server Logs Files** page displays, showing the ASP errors that have been logged.

See also:

Enabling ASP Errors Logging in this chapter

Monitoring ASP Server Performance in this chapter

Optimizing Server Performance in this chapter

Viewing Information About the ASP Server in this chapter

Viewing Server Diagnostics

You can view diagnostic information about the ASP Server from the **Server Logs Files** page of the Sun Chili!Soft ASP Administration Console, including information such as when ASP engines were started and stopped, and configuration changes that have been made since Sun Chili!Soft ASP was installed.

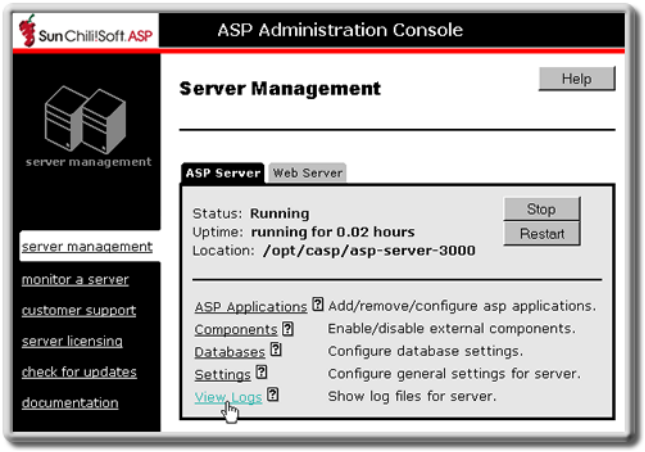
To view server diagnostics

1. Open the Administration Console by using the following URL:

   http://[HOSTNAME]:[PORT]

   where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

2. On the **ASP Server** tab of the **Server Management** page (the first page to display when you open the Administration Console), click **View Logs**.



   The **Server Logs Files** page displays.

3. Click the **Server Diagnostics** tab.

   Server diagnostic information displays.

See also:

Enabling ASP Errors Logging in this chapter

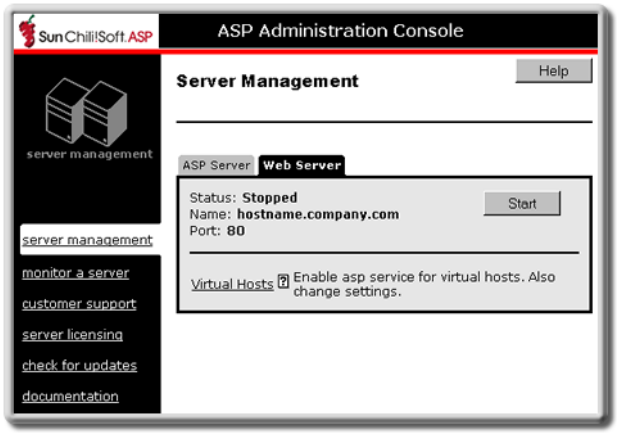
Monitoring ASP Server Performance in this chapter

Optimizing Server Performance in this chapter

Viewing Information About the ASP Server in this chapter

Managing the Web Server

Sun Chili!Soft ASP is configured to run with a Web server, which receives page requests and transfers them to the ASP Server for processing. Most Web server management is handled through the Web server's own management interface. However, from the Sun Chili!Soft ASP Administration Console **Web Server** page, you can view information about the Web server, start and stop the Web server, and enable ASP processing for individual virtual hosts.



Note

On Cobalt systems, the Cobalt Administration Console is used to manage the Web server and configure virtual hosts.

In this section:

- Starting and Stopping the Web Server
- Starting the Apache Web Server in SSL Mode
- Enabling ASP for a Virtual Host

See also:

Changing the Web Server after Installation in "Chapter 2: Installing and Uninstalling Sun Chili!Soft ASP"

Starting and Stopping the Web Server

By using the Sun Chili!Soft ASP Administration Console, you can start, stop, and restart the Web server with which the Sun Chili!Soft ASP Server is configured to run. You can also view the status of the Web server (stopped or running).

Note

This feature is not available on Cobalt systems. For those systems the Web server is managed from the Cobalt Administration Console.

To start, stop, and restart the Web Server

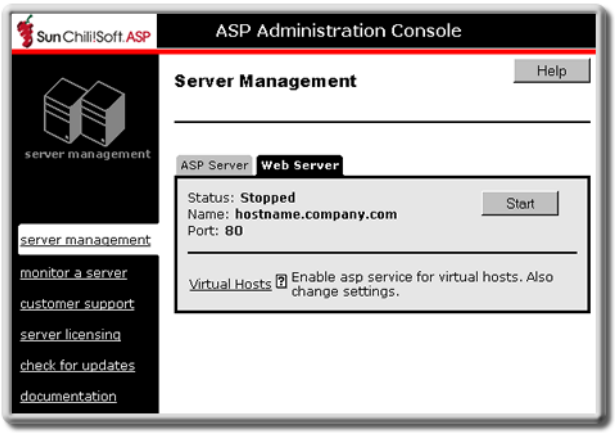
1. Open the Administration Console by using the following URL:

   http://[HOSTNAME]:[PORT]

   where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

2. On the **Server Management** page (the first page to display when you open the Administration Console), click the **Web Server** tab.

   The **Web Server** tab displays.



3. Click **Start**, **Stop,** or **Restart** as desired.

See also:

Managing the Web Server in this chapter

Starting the Apache Web Server in SSL Mode

The Sun Chili!Soft ASP Administration Console can be used to start, stop, and restart the Web server with which the Sun Chili!Soft ASP Server is configured to run. Use the following procedure to start the Apache Web server in SSL mode when Apache is started from the Sun Chili!Soft ASP Administration Console.

Note

The steps in this procedure are based on the assumption that the Apache Web server has been correctly configured with SSL support.

To start the Apache Web server in SSL mode

In the .installed_db file in the CHILI_INSTALL_DIR directory, make the following changes:

- Change:

  ```
  webserver_start_script=/<ASP_INSTALL_DIR>/INSTALL/apachectl
  "binary=/<APACHE_INSTALL_DIR>/bin/httpd"
  "conf=/<APACHE_INSTALL_DIR>/conf/httpd.conf" start
  ```

  To:

  ```
  webserver_start_script=/<APACHE_INSTALL_DIR>/bin/apachectl
  startssl
  ```

- Change:

  ```
  webserver_stop_script=/<ASP_INSTALL_DIR>/INSTALL/apachectl
  "binary=/<APACHE_INSTALL_DIR>/bin/httpd"
  "conf=/<APACHE_INSTALL_DIR>/conf/httpd.conf" stop
  ```

  To:

  ```
  webserver_stop_script=/<APACHE_INSTALL_DIR>/bin/apachectl
  stop
  ```

Enabling ASP for a Virtual Host

For the ASP Server to recognize and process an ASP application when a user requests an ASP page, you must first define the application. In a Web hosting environment that makes use of virtual hosts, such as with an Internet Service Provider (ISP), you define ASP applications in a different manner than is described in "Adding an ASP Application" in this chapter. This is because the ASP Server automatically recognizes ASP applications for each virtual host defined for the Web server. This topic describes how to use the Sun Chili!Soft ASP Administration Console to selectively enable or disable ASP processing for each virtual host.

In this scenario, the ASP application files must be located in the document root directory of the Web server or virtual host. In addition, the directory containing the global.asa file cannot be below the top-level directory of the Web server or virtual host document root. For more information about ASP applications and the global.asa file, see "Configuring ASP Applications" in this chapter.

Note

On Cobalt systems, the Cobalt Administration Console is used to manage the Web server and configure virtual hosts.

Use the following procedure to selectively enable or disable ASP processing for a virtual host.

To enable or disable ASP processing for a virtual host

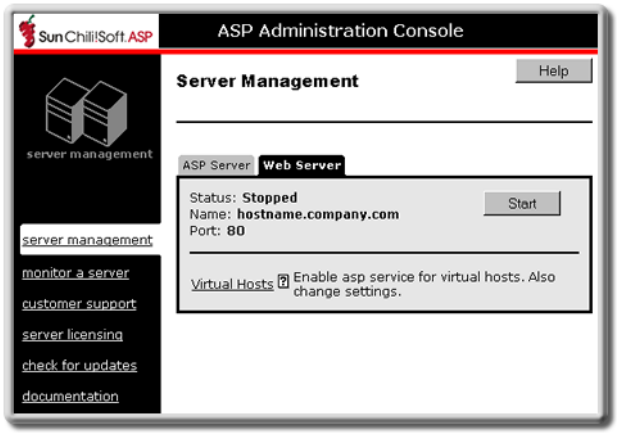
1. Open the Administration Console by using the following URL:

   http://[HOSTNAME]:[PORT]

   where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

2. On the **Server Management** page (the first page to display when you open the Administration Console), click the **Web Server** tab.

   The **Web Server** tab displays.



3. Click **Virtual Hosts**.
4. Select or deselect the check box of each virtual host for which you want to enable or disable ASP processing.
5. Click **server management** in the left navigation pane.
6. Restart the ASP Server by clicking **Restart**.

Note

Restarting the ASP Server resets all **Session** and **Application** variables.

See also:

Managing the Web Server in this chapter

Starting and Stopping the Web Server in this chapter

Enabling FrontPage Publishing

Sun Chili!Soft ASP supports Microsoft FrontPage Server Extensions. With FrontPage Server Extensions, Web authors and developers working on Windows-based computers can use the FrontPage client to publish Web pages and applications to UNIX- or Linux-based Web servers, or to Windows-based computers running a Web server other than Internet Information Server (IIS).

Sun Chili!Soft ASP supports but no longer installs FrontPage Server Extensions. You must obtain the extensions from Microsoft. Once the extensions are installed, you must take additional steps to enable users to publish their pages to the server. These issues are addressed in the following topics:

- Installing FrontPage 2002 Server Extensions
- Installing FrontPage Support on Apache 1.3.19
- Enabling FrontPage Authoring
- Setting up FrontPage Users

Sun Chili!Soft ASP enables you to run ASP pages generated by Microsoft FrontPage. Specific questions about the installation, configuration, and use of FrontPage and FrontPage Server Extensions should be directed to Microsoft or its representatives.

When publishing ASP pages created with FrontPage, be aware that **EnableParentPaths** is set to **No** by default in the Sun Chili!Soft ASP configuration file. With this configuration, **CreateObject** ("**Scripting.FileSystemObject**") calls generated in the global.asa file by FrontPage will not work. This means that you must either change **EnableParentPaths** to **Yes** (using the Administration Console) or else ASP developers must change the code that FrontPage generated in the global.asa file to **Server.CreateObject** ("**Scripting.FileSystemObject**"). However, be aware that changing this setting from the default can create a security risk for your server. For more information, see "Configuring File System Access" in this chapter.

See also:

Managing the Web Server in this chapter

Installing FrontPage 2002 Server Extensions

FrontPage Server Extensions are no longer installed with Sun Chili!Soft ASP. You must obtain the extensions from Microsoft. The FrontPage 2002 Server Extensions download is located at:

http://msdn.microsoft.com/library/en-us/dnservext/html/fpse02unix.asp

After you install FrontPage 2002 Server Extensions, you can connect to the Web server from the FrontPage 2002 client to change your administrator password ("root" by default) and set up usernames, passwords, and directories, as described in "Setting up FrontPage Users" in this chapter.

Note

To install FrontPage support on Apache 1.3.19, see "Installing FrontPage Support on Apache 1.3.19" in this section.

See also:

Enabling FrontPage Authoring in this chapter

Using the FrontPage Services File in a Shared Environment in this chapter

Installing FrontPage Support on Apache 1.3.19

This topic describes how to install FrontPage 2002 support on Apache 1.3.19. If you installed the preconfigured ("bundled") Apache 1.3.19 during the installation of Sun Chili!Soft ASP, follow the steps in "To install FrontPage support on the bundled Apache 1.3.19" in this topic. Otherwise, follow the steps immediately below.

To install FrontPage support on Apache 1.3.19

1. Uncompress and untar the Apache 1.3.19 package.

2. Download the following FrontPage files from http://msdn.microsoft.com/library/en-us/dnservext/html/fpse02unix.asp:

   `fp_install.sh`

   The FrontPage package for your platform (for example, `fp50.solaris.tar.Z`)

3. Uncompress and untar the FrontPage package.
4. Copy `[FRONTPAGE_SOURCE_DIR]/fp-patch-apache_1.3.19` to the Apache 1.3.19 source directory.
5. Type:

   ```
   #> cd [APACHE_1.3.19_SOURCE_DIR]
   ```

6. (**Note:** Make sure you use the GNU version of the "patch" program for this step.) Type:

   ```
   #> patch -p0 <fp-patch-apache_1.3.19
   ```

7. Type:

   ```
   #> ./configure --prefix=[APACHE_INSTALL_DIR] --enable-
   module=so --add-module=mod_frontpage.c
   ```

8. Type:

   ```
   #> make
   ```

9. Type:

   ```
   #> make install
   ```

10. In the Apache httpd.conf file, change the `AllowOverride` option of the document root directory to something other than "None" (for example, "All").
11. Run the `fp_install.sh` script.

    Note: During the FrontPage install, you can skip the step for configuring subwebs.

12. Install Sun Chili!Soft ASP.

To install FrontPage support on the bundled Apache 1.3.19

1. Run the Sun Chili!Soft ASP installer and install the bundled Apache 1.3.19.
2. Download the following FrontPage files from http://msdn.microsoft.com/library/en-us/dnservext/html/fpse02unix.asp:

   `fp_install.sh`

   The FrontPage package for your platform (for example, `fp50.solaris.tar.Z`)

3. Run the `fp_install.sh` script.

Enabling FrontPage Authoring

With Sun Chili!Soft ASP, Web authors can publish their work to the Web server using the FrontPage client. To enable this capability, FrontPage Server Extensions must be installed and FrontPage authoring must be enabled on the Web server. For information about enabling FrontPage authoring, consult your FrontPage documentation.

FrontPage Server Extensions are supported but not installed by Sun Chili!Soft ASP. You must obtain the extensions from Microsoft, as described in "Installing FrontPage 2002 Server Extensions" in this chapter.

Once you have enabled FrontPage authoring, you can then access the FrontPage root Web on your Web server to configure it for FrontPage users, as described in "Setting up FrontPage Users" in this chapter.

Note

On Cobalt systems, FrontPage Server Extensions are managed from the Cobalt Administration Console.

See also:

Enabling FrontPage Publishing in this chapter

Installing FrontPage 2002 Server Extensions in this chapter

Setting up FrontPage Users in this chapter

Using the FrontPage Services File in a Shared Environment in this chapter

Setting up FrontPage Users

Before you can set up directories and Webs for FrontPage users, you must first install and configure FrontPage Server Extensions, as described in "Installing FrontPage 2002 Server Extensions" in this chapter. Then you must enable FrontPage authoring on the Web server, as described in your Microsoft product documentation.

Once you have done this, you can use the FrontPage client to connect to the FrontPage root Web to configure it for users. You can change your FrontPage administrator password ("root" by default) and set up usernames, passwords, and directories (called "Webs") in FrontPage. For more information about changing the administrator password and adding users and user Webs, see your Microsoft product documentation.

To connect to the FrontPage root Web

1. Make sure FrontPage Server Extensions are installed and FrontPage authoring is enabled on the Web server.
2. From the FrontPage client **File** menu, click **Open Web**.
3. In the **Folder Name** box, type the following URL:

   http://[HOSTNAME]

where [HOSTNAME] is the hostname of your Web server.

4. Click **Open**.
5. When prompted, type "admin" for the username and "root" for the password.
6. Make the desired configuration changes, as described in the FrontPage documentation.

See also:

Enabling FrontPage Publishing in this chapter

Using the FrontPage Services File in a Shared Environment in this chapter

Configuring a Database

Sun Chili!Soft ASP enables ASP developers to connect with several types of databases from within an ASP application. It provides the built-in ADO **Connection** object that developers can use to initiate a database connection, along with a set of ODBC drivers that enable the ASP Server and ODBC Manager to establish and maintain the connection. (For more information about creating and initializing ADO database connections from within an ASP application, see "Connecting to a Database" in "Chapter 4: Building a Sun Chili!Soft ASP Application.")

For UNIX and Linux versions of Sun Chili!Soft ASP, the setup program automatically installs the ODBC drivers for a number of databases (ODBC drivers are not installed with Sun Chili!Soft ASP for Windows). However, for some types of databases, the system administrator must take additional steps to configure the ASP Server. For example, the system administrator must configure SequeLink to enable connections from an ASP Server running on a UNIX or Linux system to a Microsoft Access and Microsoft SQL Server 6.5 database running on a Windows system. In addition, the system administrator might want to create system DSNs to make it easier for developers to connect with databases. This section describes how to create and edit DSNs, and how to configure the ASP Server to connect with supported databases.

Note

> You can view the list of installed drivers from the Administration Console, as described in "Viewing the List of ODBC Drivers" in this chapter. Sun Chili!Soft provides customer support only for ODBC drivers that are installed with Sun Chili!Soft ASP.

In this section:

- Configuring Data Source Names (DSNs)
- Viewing the List of ODBC Drivers
- Configuring SequeLink
- Configuring the Database Environment
- Configuring Database Parameters

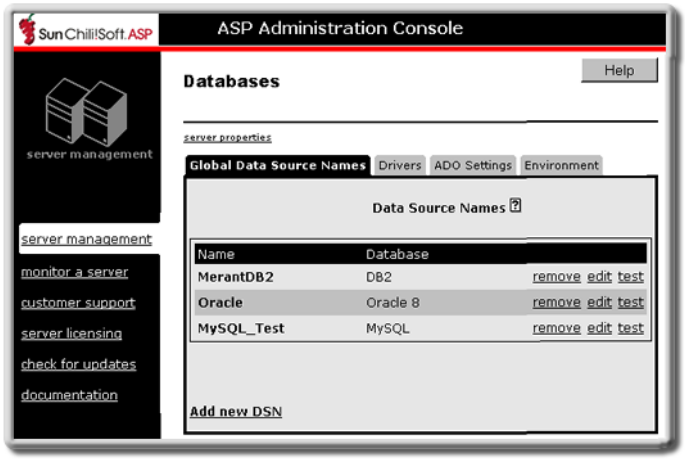
See also:

Creating Database Connections on the Server in "Chapter 2: Installing and Configuring Sun Chili!Soft ASP"

Configuring Data Source Names (DSNs)

To make it easier for developers to connect an ASP application to a database, the system administrator can add a system data source name (DSN) to the ASP Server. DSNs store information about a database that the ASP Server and ODBC Manager use for connecting to it. Developers can use the system DSN in connection strings on ASP pages to incorporate database information by reference, rather than specifying the complete set of information in each string.

You can access DSN configuration settings on the **Data Source Names** tab of the Sun Chili!Soft ASP Administration Console **Databases** page. This tab displays the list of system DSNs that are currently configured for the ASP Server. It also provides access to settings for adding a new DSN to the ASP Server, and for removing, editing, and testing an existing DSN.



Note

To protect the security of your database, in a shared Web hosting environment, you might prefer that developers use DSN-less connection strings or file DSNs instead of system DSNs. For a discussion of these security issues, see "Enabling Database Connections on the Server" in "Chapter 2: Installing and Configuring Sun Chili!Soft ASP."

For Windows systems, DSNs are created and managed from the Windows Control Panel. See Microsoft documentation for more information about using the Windows ODBC Data Source Administrator.

The topics in this section describe how to add, remove, edit, and test system DSNs. In addition to the steps described in this section, you also might need to take other steps to configure the ASP Server to support a particular database. For more information, see "Configuring a Database" in this chapter.

In this section:

- Adding a DSN

- Removing a DSN
- Editing a DSN
- Testing a DSN

Adding a DSN

To add a system DSN to the ASP Server, use the following procedure. When you add a DSN, Sun Chili!Soft ASP automatically sets the correct parameters for the databases. If necessary, you can edit these parameters, as described in "Editing a DSN" in this chapter.

Notes about specific databases

Microsoft Access and Microsoft SQL Server 6.5: Sun Chili!Soft ASP includes an ODBC driver for Microsoft SQL Server 7.0 and 2000, but you must use SequeLink 4.51a for connecting to Microsoft Access and Microsoft SQL Server 6.5 databases. You can create a DSN for SequeLink using the procedure in this topic. However, before you do this, you must first take the steps described in "Configuring SequeLink" in this chapter.

Oracle and Informix: For these databases, after adding a DSN you must also define database environment variables, as described in "Configuring the Database Environment" in this chapter. Unless the environment variables have been set previously, after you finish adding a new DSN, the Administration Console opens the appropriate page on which to configure these settings.

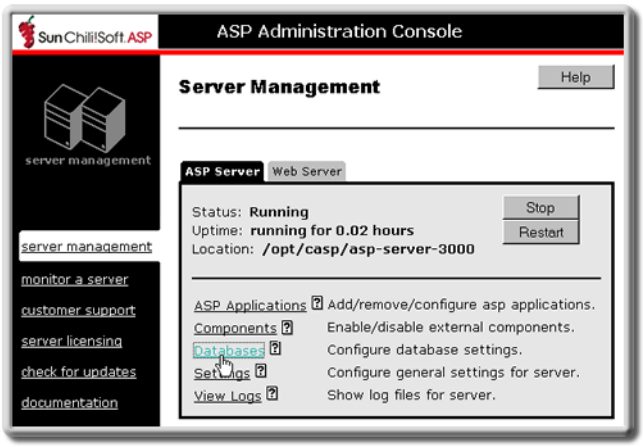
To add a system DSN

1. Open the Administration Console by using the following URL:

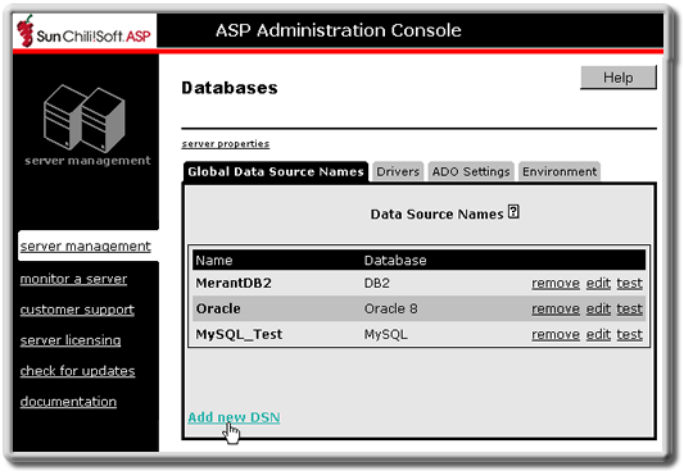
   http://[HOSTNAME]:[PORT]

   where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

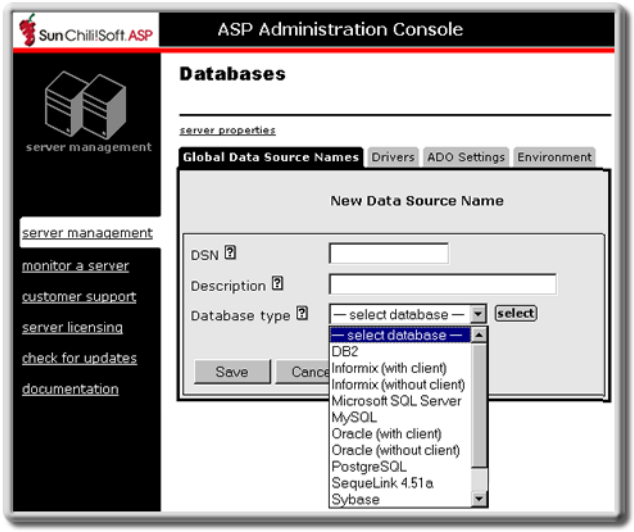
2. On the **ASP Server** tab of the **Server Management** page (the first page to display when you open the Administration Console), click **Databases**.



3. On the **Databases** page, click **Add new DSN**.



The **New Data Source Name** page displays.



4. In the **DSN** box, type a name for the DSN.
5. If desired, in the **Description** box, type a description of the DSN to help distinguish it from other DSNs.
6. In the **Database type** drop-down list, select the type of database for which you want to configure a DSN (for Microsoft Access and Microsoft SQL Server 6.5 databases, select **SequeLink 4.51a**).
7. In the remaining text boxes, provide the requested information. For more information, see the topic for your database in "Configuring Database Parameters" in this chapter.
8. To save your changes, click **Save**, and then click **Done**.

   – or –

   Click **Cancel** to revert to the last settings that were saved.

The new DSN displays in the **Data Source Names** list. After adding a DSN, it is a good idea to test it, as described in "Testing a DSN" in this chapter.

Note

For Windows systems, data source names are created and managed from the Windows Control Panel. See Microsoft documentation for more information about using the Windows ODBC Data Source Administrator.

See also:

Configuring Data Source Names (DSNs) in this chapter

Removing a DSN in this chapter

Editing a DSN in this chapter

Enabling Database Connections on the Server in "Chapter 2: Installing and Configuring Sun Chili!Soft ASP"

Removing a DSN

You remove a system DSN from the ASP Server by using the Sun Chili!Soft ASP Administration Console. When you do this, it no longer can be used in an ASP application to reference database connection information.

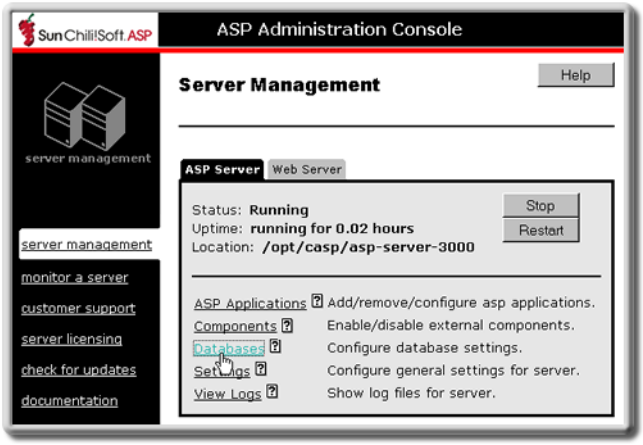
To remove a system DSN

1. Open the Administration Console by using the following URL:

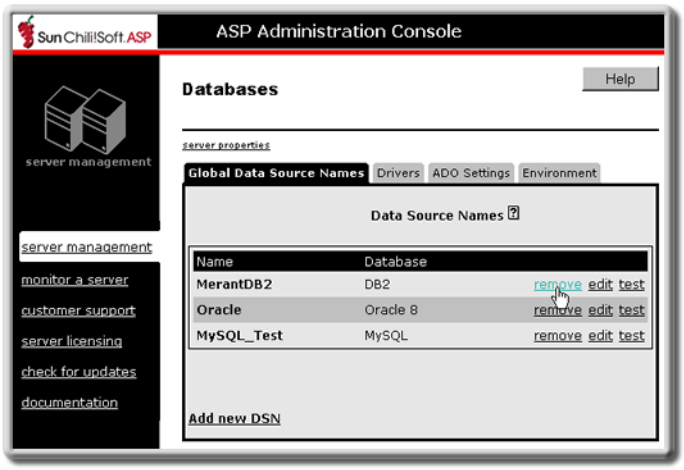
   http://[HOSTNAME]:[PORT]

   where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

2. On the **ASP Server** tab of the **Server Management** page (the first page to display when you open the Administration Console), click **Databases**.



3. On the **Databases** page, in the same line as the name of the DSN you want to remove, click **remove**.



4. When prompted to confirm the removal, click **Yes**, and then click **Done**.

See also:

Configuring Data Source Names (DSNs) in this chapter

Adding a DSN in this chapter

Editing a DSN in this chapter

Testing a DSN in this chapter

Enabling Database Connections on the Server in "Chapter 2: Installing and Configuring Sun Chili!Soft ASP"

Editing a DSN

After adding a system DSN to the ASP Server you can change its information, such as name, description, IP address, username, and password. You can also add values for parameters that were not configured when you added the DSN.

When you add a new DSN to the ASP Server, Sun Chili!Soft ASP automatically sets the correct parameters for the database. Changing these parameters can affect database performance and it is not recommended that you do so. The default settings are sufficient for most applications. Before editing database parameters, see the topics describing the required parameters for each ODBC driver in "Configuring Database Parameters" in this chapter.

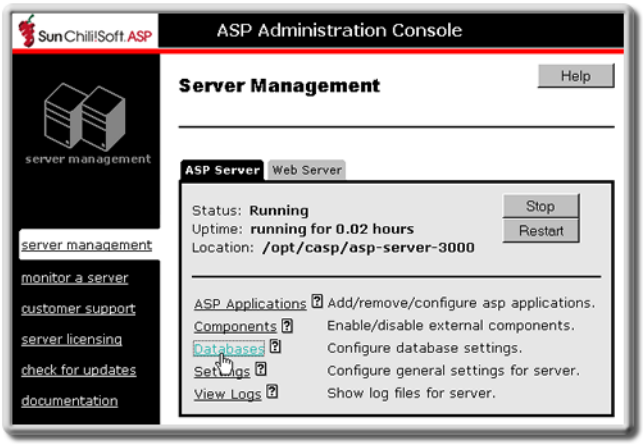
To edit system DSN information

1. Open the Administration Console by using the following URL:

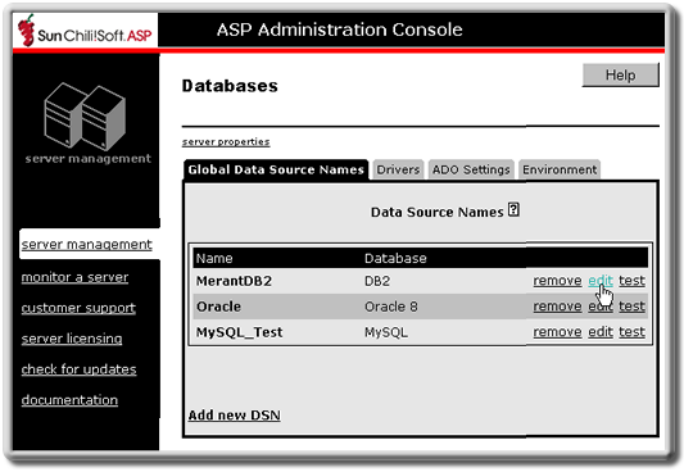
   http://[HOSTNAME]:[PORT]

   where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

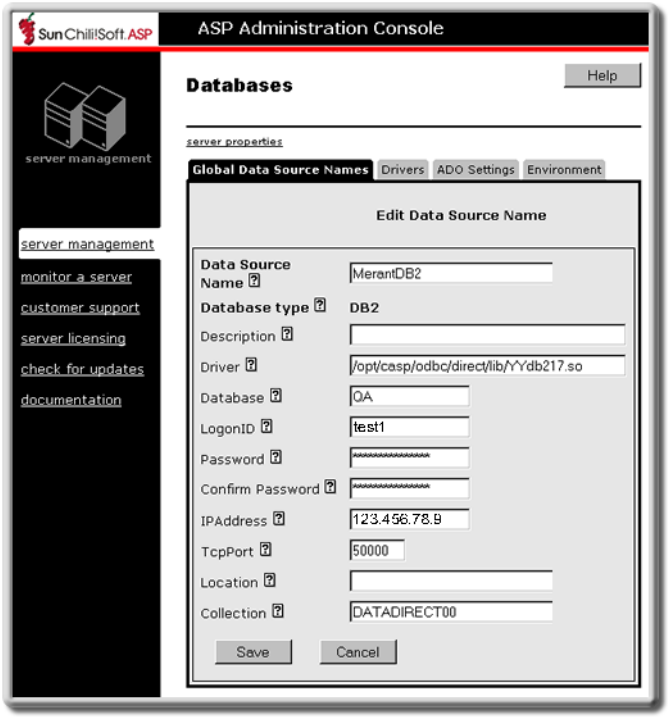
2. On the **ASP Server** tab of the **Server Management** page, click **Databases**.



3. On the **Databases** page, in the same line as the name of the DSN you want to change, click **edit**.



The **Edit Data Source Name** page displays.



4. Change the database parameters as desired.
5. To save your changes, click **Save** and then click **Done**.

   – or –

   Click **Cancel** to revert to the last settings that were saved.

See also:

Configuring Data Source Names (DSNs) in this chapter

Adding a DSN in this chapter

Removing a DSN in this chapter

Testing a DSN in this chapter

Testing a DSN

When you have finished adding a new system DSN or editing its parameters, you can use the following procedure to verify that it is functioning correctly.

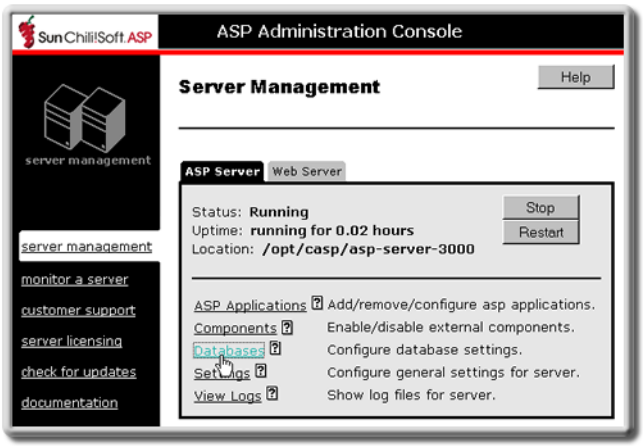
To test a system DSN

1. Open the Administration Console by using the following URL:

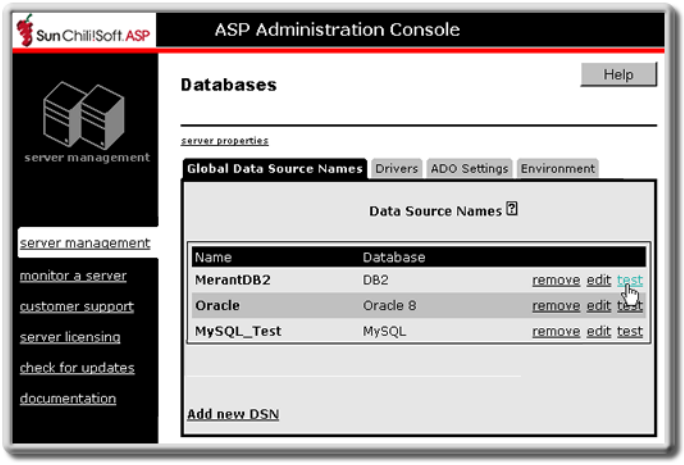
   http://[HOSTNAME]:[PORT]

   where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

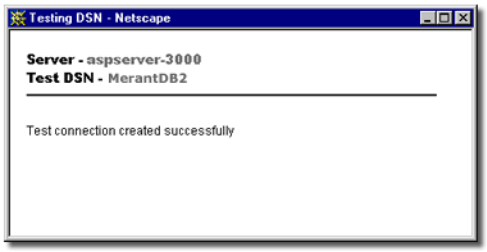
2. On the **ASP Server** tab of the **Server Management** page (the first page to display when you open the Administration Console), click **Databases**.



3. On the **Databases** page, in the same line as the name of the DSN you want to change, click **test**.



A dialog box opens, displaying information about the connection. If it reveals an error, correct the problem and then retest the connection.



Note

If you are running Sun Chili!Soft ASP with Zeus Web Server, and the test fails, it could be because the Web server hostname is not configured correctly. When you configure Zeus Web Server to run with Sun Chili!Soft ASP, you should NOT use the fully qualified domain name ("server address") for the Web server hostname. Instead, use the hostname alone. For example, do not use this: hostname.domain.com. Instead, use this: hostname.

If your Zeus Web server is configured with the server address rather than the hostname alone, you can test a DSN by using the SQLEXECUTE diagnostic included with Sun Chili!Soft ASP. For more information, see "Accessing Documentation, Samples, and Diagnostics" in "Introduction: About This Documentation."

See also:

Configuring Data Source Names (DSNs) in this chapter

Adding a DSN in this chapter

Editing a DSN in this chapter

Removing a DSN in this chapter

Viewing the List of ODBC Drivers

Sun Chili!Soft ASP enables you to connect to a variety of ODBC-compliant databases by using the appropriate ODBC driver. To verify whether Sun Chili!Soft ASP supports a specific version of a database, you can view the list of ODBC drivers included with Sun Chili!Soft ASP on the **Drivers** tab of the Sun Chili!Soft ASP Administration Console **Databases** page.

Note

Sun Chili!Soft ASP for UNIX and Linux installs the ODBC drivers for a number of databases. Sun Chili!Soft ASP for Windows does not install any ODBC drivers. For Windows systems, the list of installed ODBC drivers can be viewed from the Windows Control Panel (double-click the **Data Sources** icon). See Microsoft documentation for more information about using the Windows ODBC Data Source Administrator.

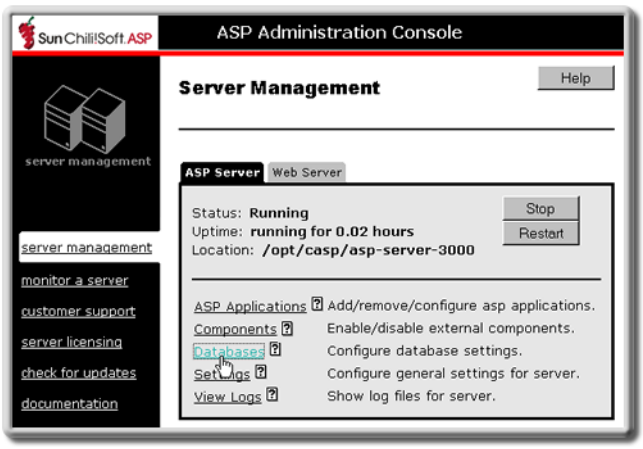
To view the list of ODBC Drivers

1. Open the Administration Console by using the following URL:

   http://[HOSTNAME]:[PORT]

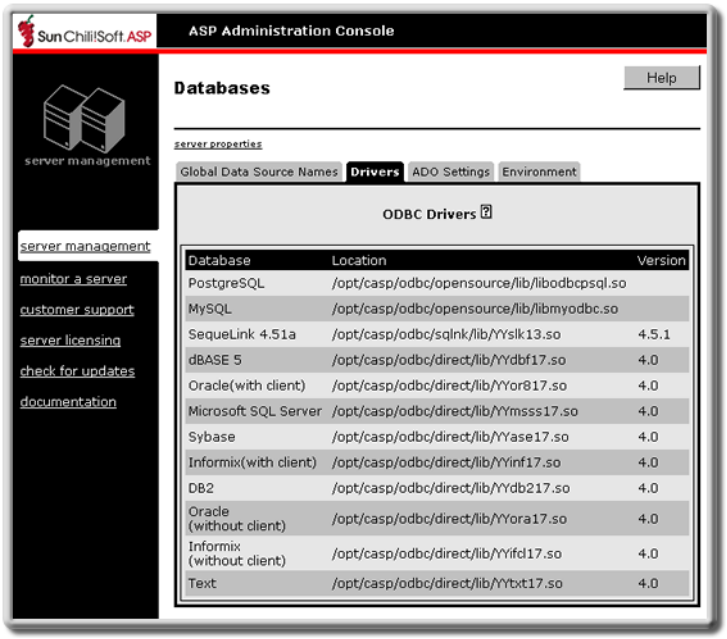
   where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

2. On the **ASP Server** tab of the **Server Management** page (the first page to display when you open the Administration Console), click **Databases**.



3. Click the **Drivers** tab.

   The **ODBC Drivers** page displays, showing the list of installed ODBC drivers and their locations in the file system.



Configuring SequeLink

Sun Chili!Soft ASP includes the client portion of DataDirect SequeLink 4.5.1, which enables you to connect to a remote Microsoft Access or Microsoft SQL Server 6.5 database running on Windows 95, Windows 98, Windows NT 3.51 or 4.0, or Windows 2000. The SequeLink client resides on the same computer as the ASP Server and behaves like an ODBC driver. It communicates with a SequeLink server running on the remote database server.

Before you can use SequeLink to connect to a remote database, you must take the following steps:

1. Configure the SequeLink client software for Microsoft Access or Microsoft SQL Server 6.5, as described later in this topic.
2. Install and configure the SequeLink server software on the database server. You can download the software from the Sun Chili!Soft Web site at:

   ftp://ftp.chilisoft.com/chiliasp/sequelink/slkntsrv.zip
3. Add a SequeLink DSN by using the Administration Console, as described in "Adding a DSN" in this chapter. When you do this, be sure to use the DSN name that you create in the following procedure.

To configure the SequeLink client for Microsoft Access

1. From the Sun Chili!Soft ASP instance subdirectory of the Sun Chili!Soft ASP installation directory, type the following command:

   ```
   ./setsqlnk
   ```
2. Select **[2] New** to create a new SequeLink DSN. Enter the following information when prompted:

   Name: The name of the SequeLink DSN that you want to create (you must use the same DSN name when you configure the SequeLink DSN in the Administration Console).

   Description: Optional.

   Transliteration: Do not configure or change this option.

   Select a network: Select **TCP**.

   Host: Enter the IP address of the Windows server on which your database is running.

   ServerType: Select **Windows NT Server**.

   User: Enter the username for accessing the Windows server.

   Password: Enter the password for accessing the Windows server.

   Select a Database service: Select **ODBC MS Access**.

   Name: Enter the name of the Windows service that you created when installing the SequeLink Server software.

   Database: Enter the name of the database to which you want this DSN to connect, for example, E:\data\publish.mdb (note the Windows syntax for providing the path information).
3. Select **[6] Test**. The setup utility should return **Test Passed**. If you receive an error, select **[4]** to edit your DSN information.
4. After completing SequeLink setup, select **[0]** to exit.
5. Configure the SequeLink DSN, as described in "Adding a DSN" in this chapter.

To configure the SequeLink client for Microsoft SQL Server 6.5

1. From the Sun Chili!Soft ASP instance subdirectory of the Sun Chili!Soft ASP installation directory, type:

   ```
   ./setsqlnk
   ```

2. Select **[2] New** to create a new SequeLink DSN. Enter the following information when prompted:

   Name: The name of the SequeLink DSN that you want to create (you must use the same DSN name when you configure the SequeLink DSN in the Administration Console).

   Description: Optional.

   Transliteration: Do not configure or change this setting.

   Select a network: Select **TCP**.

   Host: Enter the IP address of the Windows server on which your database is running.

   ServerType: Select **Windows NT Server**.

   User: Enter the username for accessing the Windows server.

   Password: Enter the password for accessing the Windows server**.**

   Select a Database service: Select **MS SQL Server**.

   Name: Enter the name of the Windows service that you created when installing the SequeLink Server software.

   Database: Enter the name of the database to which you want this DSN to connect.

   User: Enter the SQL Server username for accessing the database.

   Password: Enter the password for accessing the SQL Server database (required).

3. Select **[6] Test**. The setup utility should return **Test Passed**. If you receive an error, select **[4]** to edit your DSN information.
4. After completing SequeLink setup, select **[0]** to exit.
5. Configure the SequeLink DSN, as described in "Adding a DSN" in this chapter.

See also:

Configuring a Database in this chapter

SequeLink Parameters in this chapter

Configuring the Database Environment

When you configure data source names (DSNs) for Oracle and Informix databases for Sun Chili!Soft ASP, you must also specify additional environment information, as described in this section. For more information about the settings to use, consult your database administrator. For more information about configuring DSNs, see "Configuring Data Source Names (DSNs)" in this chapter.

In this section:

- Setting Oracle Environment Variables
- Setting Informix Environment Variables

Setting Oracle Environment Variables

When you configure a DSN for an Oracle database, you must also specify values for the **Oracle_Home** and **Library path** environment variables by using the following procedure. For information about the values to set for these variables, consult your database administrator.

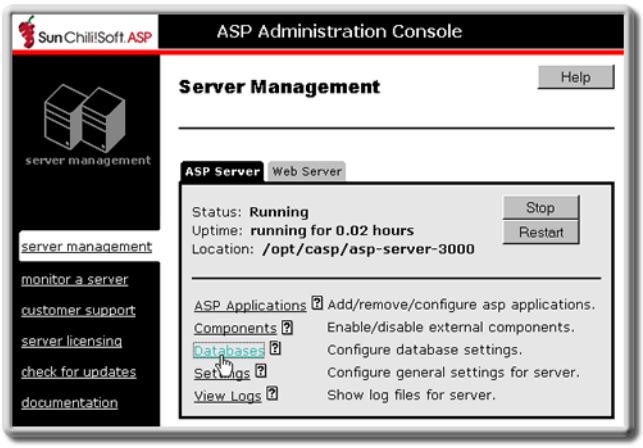
To set Oracle environment variables

1. Open the Administration Console by using the following URL:

   http://[HOSTNAME]:[PORT]

   where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

2. On the **ASP Server** tab of the **Server Management** page (the first page to display when you open the Administration Console), click **Databases**.



3. On the **Databases** page, click the **Environment** tab.

   The **Environment Database Specific Variables** page displays.

4. Under the **Oracle** heading, in the **ORACLE_HOME** and **Library path** boxes, type the desired values.
5. Select the **Restart the ASP Server after saving** check box, and then click **Save**.

Note

Restarting the ASP Server resets all **Session** and **Application** variables.

See also:

Configuring a Database in this chapter

Setting Informix Environment Variables

When you configure a DSN for an Informix 7 or 9 database, you must also specify values for the **INFORMIXDIR**, **INFORMIXSERVER**, **Temp path**, **Library path**, and **eSQL path** environment variables by using the following procedure. For information about the values to set for these variables, consult your database administrator.

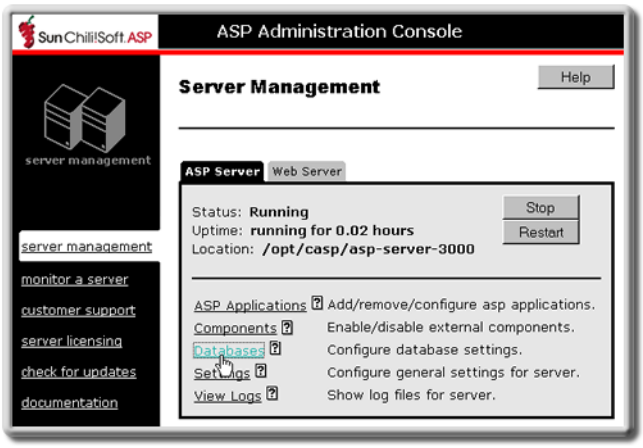
To set Informix environment variables

1. Open the Administration Console by using the following URL:

   http://[HOSTNAME]:[PORT]

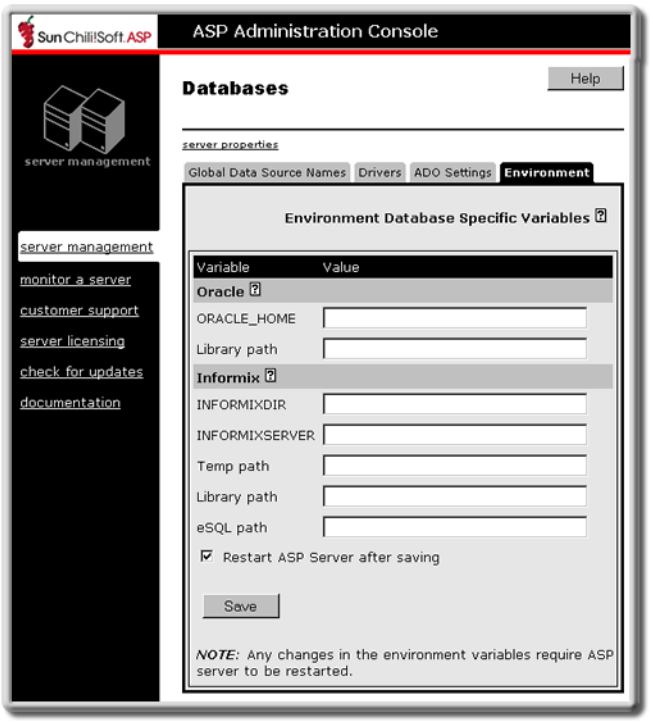
   where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

2. On the **ASP Server** tab of the **Server Management** page (the first page to display when you open the Administration Console), click **Databases**.



3. On the **Databases** page, click the **Environment** tab.

   The **Environment Database Specific Variables** page displays.



4. Under the **Informix** heading, in the **INFORMIXDIR**, **INFORMIXSERVER**, **Temp path**, **Library path**, and **eSQL path** boxes, type the desired values.

5. Select the **Restart the ASP Server after saving** check box, and then click **Save**.

Note

Restarting the ASP Server resets all **Session** and **Application** variables.

See also:

Configuring a Database in this chapter

Configuring Database Parameters

To make it easier for Web developers to connect to a database from an ASP page, you can "add" a system DSN for the database to the ASP Server, as described in "Adding a DSN" in this chapter. When you do this, Sun Chili!Soft ASP automatically configures the appropriate parameters for the ODBC driver installed for that database. The ASP Server and ODBC Manager use this information to establish the connection.

In most cases, you should not change the parameters that Sun Chili!Soft ASP configures. However, there might be times when you need to edit them. This topic provides reference information about the parameters that are configured for the ODBC drivers installed by Sun Chili!Soft ASP.

Note

Sun Chili!Soft ASP for UNIX and Linux installs the ODBC drivers for a number of databases (ODBC drivers are not installed with Sun Chili!Soft ASP for Windows). You can view the list of installed drivers from the Administration Console, as described in "Viewing the List of ODBC Drivers" in this chapter. Sun Chili!Soft provides support only for ODBC drivers that are installed with Sun Chili!Soft ASP.

In this section:

- DB2
- dBASE 5
- Informix (with and without client)
- Microsoft SQL Server
- MySQL
- Oracle (with and without client)
- PostgreSQL
- SequeLink (for connecting to Microsoft Access and Microsoft SQL Server 6.5 databases)
- Sybase
- Text

See also:

Configuring a Database in this chapter

Editing a DSN in this chapter

DB2 Parameters

The following table describes the DB2 (UDB, v7.1) database parameters available for configuring system DSNs, as they appear on the Sun Chili!Soft ASP Administration Console **New Data Source Name** and **Edit Data Source Name** pages. For more information, see "Configuring Data Source Names (DSNs)" in this chapter.

Parameter	Explanation
DSN*	This is the name of the data source name (DSN) you are configuring. It must match the catalogued name of the DB2 database.
Description	This field provides a description of the DSN to distinguish it from others.
Database type*	This indicates for which type of database you are configuring this DSN (DB2).
Driver	On the **New Data Source Name** page, this is the name of the ODBC driver installed for the type of database selected in the **Database type** box (DB2). It is a nonconfigurable field. On the **Edit Data Source Name** page, this is the absolute path name of the ODBC driver specified for this DSN. It is a configurable field.
Database*	This entry must match the catalogued name of this DB2 database. The data source name (DSN) specified above must match this entry.
LogonID*	This is the username required for accessing the database. If the username is not provided when configuring a system DSN, every connection string using this DSN must include the username. **Important Security Note:** To prevent access to a database by unauthorized users in shared Web hosting environments, it is recommended that the username and password be provided in each connection string, rather than in the system DSN.
Password*	This is the password required for accessing the database. If the password is not provided when configuring a system DSN, every connection string using the DSN must include the password. **Important Security Note:** To prevent access to a database by unauthorized users in shared Web hosting environments, it is recommended that the username and password be provided in each connection string, rather than in the system DSN.

IPAddress*	This is the IP address for the database server (DB2).
TcpPort*	This is the port for the database server (DB2).
Location	Specify this attribute only if the DB2 database is running on OS/390. Location is a path that specifies the DB2 location name. Use the name that was defined during the local DB2 installation.
Collection	Specify this attribute only if the DB2 database is running on OS/390. Collection is the name that identifies a group of packages. These packages include the Connect ODBC for DB2 Wire Protocol driver packages. The default is DATADIRECTOO.
Package	This is the package created by the DataDirect driver that reflects all of the parameters associated with a specific database (the parameters you specified). **Package** is displayed in the Administration Console only when you are editing an existing DSN, not adding a new one. **Note:** Do not edit this package. The package is unique to a specific database.

*Required parameters

See also:

Configuring a Database in this chapter

Configuring Database Parameters in this chapter

dBASE 5 Parameters

The following table describes the dBASE 5 database parameters available for configuring system DSNs, as they appear on the Sun Chili!Soft ASP Administration Console **New Data Source Name** and **Edit Data Source Name** pages. For more information, see "Configuring Data Source Names (DSNs)" in this chapter.

Parameter	Explanation
DSN*	This is the name of the data source name (DSN) you are configuring.
Description	This field provides a description of the DSN to help distinguish it from others.
Database type*	This indicates for which type of database you are configuring this DSN (dBASE 5).
Driver*	On the **New Data Source Name** page, this is the name of the ODBC driver installed for the type of database selected in the **Database type** box (dBASE 5). It is a nonconfigurable field.

	On the **Edit Data Source Name** page, this is the absolute path name of the ODBC driver specified for this DSN. It is a configurable field.
Database*	This is the path name of the directory in which the DBF files reside.
IntlSort	This field determines the order in which records are retrieved when you issue a **Select** statement with an **Order By** clause. When set to **0** (the default), ASCII sort order is used. Items are sorted alphabetically, with uppercase letters preceding lowercase letters (for example, "A, b, C" would be sorted as "A, C, b"). When set to **1**, international sort order is used, as defined by your operating system. The order is always alphabetic, regardless of case.

***** Required parameters

See also:

Configuring a Database in this chapter

Configuring Database Parameters in this chapter

Informix Parameters (with Client): UNIX only

The following table describes the Informix 7 or 9 (with client) database parameters available for configuring system DSNs, as they appear on the Sun Chili!Soft ASP Administration Console **New Data Source Name** and **Edit Data Source Name** pages. For more information, see "Configuring Data Source Names (DSNs)" in this chapter. For Informix Parameters (without Client), see below.

Note

This driver is not installed with Sun Chili!Soft ASP for Linux. Parameters for the Informix driver that is installed with Sun Chili!Soft ASP for Linux are listed below in Informix Parameters (without Client).

Parameter	Explanation
DSN*	This is the name of the data source name (DSN) you are configuring.
Description	This field provides a description of the DSN to help distinguish it from others.
Database type*	This indicates for which type of database you are configuring this DSN (Informix).
Driver*	On the **New Data Source Name** page, this is the name of the ODBC driver installed for the type of database selected in the **Database type** box (Informix). It is a nonconfigurable field. On the **Edit Data Source Name** page, this is the absolute path name of

	the ODBC driver specified for this DSN. It is a configurable field.
ServerName*	This is the name of the database server.
HostName*	This is the name of the computer on which the Informix server resides.
Database*	Because there can be multiple installations of a database running on one computer, each database is given its own name. This parameter indicates the name of the database for which you are configuring this DSN.
LogonID	This is the username required for accessing the database. If the username is not provided when configuring a system DSN, every connection string using this DSN must include the username. **Important Security Note:** To prevent access to a database by unauthorized users in shared Web hosting environments, it is recommended that the username and password be provided in each connection string, rather than in the system DSN.
Password	This is the password required for accessing the database. If the password is not provided when configuring a system DSN, every connection string using this DSN must include the password. **Important Security Note:** To prevent access to a database by unauthorized users in shared Web hosting environments, it is recommended that the username and password be provided in each connection string, rather than in the system DSN.

* Required parameters

Informix Parameters (without Client): UNIX and Linux

The following table describes the Informix 2000 (without client) database parameters available for configuring system DSNs, as they appear on the Sun Chili!Soft ASP Administration Console **New Data Source Name** and **Edit Data Source Name** pages. For more information, see "Configuring Data Source Names (DSNs)" in this chapter.

Note

On Linux, only one Informix driver is listed in the Sun Chili!Soft ASP Administration Console.

Parameter	Explanation
DSN*	This is the name of the data source name (DSN) you are configuring.
Description	This field provides a description of the DSN to help distinguish it from

	others.
Database type*	This indicates for which type of database you are configuring this DSN (Informix).
Driver*	On the **New Data Source Name** page, this is the name of the ODBC driver installed for the type of database selected in the **Database type** box (Informix). It is a nonconfigurable field. On the **Edit Data Source Name** page, this is the absolute path name of the ODBC driver specified for this DSN. It is a configurable field.
Database*	Because there can be multiple installations of a database running on one computer, each database is given its own name. This parameter indicates the name of the database for which you are configuring this DSN.
LogonID	This is the username required for accessing the database. If the username is not provided when configuring a system DSN, every connection string using this DSN must include the username. **Important Security Note:** To prevent access to a database by unauthorized users in shared Web hosting environments, it is recommended that the username and password be provided in each connection string, rather than in the system DSN.
Password	This is the password required for accessing the database. If the password is not provided when configuring a system DSN, every connection string using this DSN must include the password. **Important Security Note:** To prevent access to a database by unauthorized users in shared Web hosting environments, it is recommended that the username and password be provided in each connection string, rather than in the system DSN.
HostName*	This is the name of the computer on which the Informix server resides.
PortNumber*	This is the port on which the database server is configured to listen. Ask your database administrator for this information.
ServerName*	This is the name of the database server.

***** Required parameters

See also:

Configuring a Database in this chapter

Configuring Database Parameters in this chapter

Microsoft SQL Server Parameters

The following table describes the Microsoft SQL Server 7.0 and 2000 database parameters available for configuring system DSNs, as they appear on the Sun Chili!Soft ASP Administration Console **New Data Source Name** and **Edit Data Source Name** pages. For more information, see "Configuring Data Source Names (DSNs)" in this chapter.

Parameter	Explanation
Data Source Name*	This is the name of the data source name (DSN) you are configuring.
Database type*	This indicates for which type of database you are configuring this DSN (Microsoft SQL Server).
Description	This field provides a description of the DSN to help distinguish it from others.
Driver*	On the **New Data Source Name** page, this is the name of the ODBC driver installed for the type of database selected in the **Database type** box (Microsoft SQL Server). It is a nonconfigurable field. On the **Edit Data Source Name** page, this is the absolute path name of the ODBC driver specified for this DSN. It is a configurable field.
Database*	Because there can be multiple installations of a database running on one computer, each database is given its own name. This parameter indicates the name of the database for which you are configuring this DSN.
ServerIPAddress*	This is the IP address of the SQL Server 7.0 or 2000 database server.
ServerPortNumber*	This is the port on which the SQL Server 7.0 or 2000 database server is configured to listen. The default is 1433.
LogonID	This is the username required for accessing the database. If the username is not provided when configuring a system DSN, every connection string using this DSN must include the username. **Important Security Note:** To prevent access to a database by unauthorized users in shared Web hosting environments, it is recommended that the username and password be provided in each connection string, rather than in the system DSN.
Password	This is the password required for accessing the database. If the password is not provided when configuring a system DSN, every connection string using the DSN must include the password. **Important Security Note:** To prevent access to a database by unauthorized users in shared Web hosting environments, it is recommended that the username and password be provided in each connection string, rather than in the

system DSN.

* Required parameters

See also:

Configuring a Database in this chapter

Configuring Database Parameters in this chapter

MySQL Parameters (Solaris, AIX, and Linux only)

The following table describes the MySQL 3.22.30 and 3.23.49 database parameters available for configuring system DSNs, as they appear on the Sun Chili!Soft ASP Administration Console **New Data Source Name** and **Edit Data Source Name** pages. For more information, see "Configuring Data Source Names (DSNs)" in this chapter.

Parameter	Explanation
Data Source Name*	This is the name of the data source name (DSN) you are configuring.
Database type*	This indicates for which type of database you are configuring this DSN (MySQL).
Description	This field provides a description of the DSN to help distinguish it from others.
Driver*	On the **New Data Source Name** page, this is the name of the ODBC driver installed for the type of database selected in the **Database type** box (MySQL). It is a nonconfigurable field. On the **Edit Data Source Name** page, this is the absolute path name of the ODBC driver specified for this DSN. It is a configurable field.
Server*	This is the IP address of the MySQL database server. If this field is empty, the server is assumed to be running on the local computer.
Port*	This is the port on which the MySQL database server is configured to listen. The default is 3306.
Database*	Because there can be multiple installations of a database running on one computer, each database is given its own name. This parameter indicates the name of the database for which you are configuring this DSN.
User	This is the username required for accessing the database. If the username is not provided when configuring a system DSN, every connection string using this DSN must include the username. **Important Security Note:** To prevent access to a database by unauthorized users in shared Web hosting environments, it is recommended that the username and password be provided in each connection string, rather than in the

	system DSN.
Password	This is the password required for accessing the database. If the password is not provided when configuring a system DSN, every connection string using the DSN must include the password. **Important Security Note:** To prevent access to a database by unauthorized users in shared Web hosting environments, it is recommended that the username and password be provided in each connection string, rather than in the system DSN.
UseCursorLib	When this option is enabled (the check box is selected), ODBC Manager cursor support overrides ODBC driver cursor support. This enables **RecordSet.Update**, which is not supported in the default MyODBC driver. This parameter is enabled by default.

***** Required parameters

See also:

Configuring a Database in this chapter

Configuring Database Parameters in this chapter

Oracle Parameters (with Client)

The following table describes the Oracle 7 and 8.05 (with client) database parameters available for configuring system DSNs, as they appear on the Sun Chili!Soft ASP Administration Console **New Data Source Name** and **Edit Data Source Name** pages. For more information, see "Configuring Data Source Names (DSNs)" in this chapter. For Oracle Parameters (without Client), see below.

Parameter	Explanation
DSN*	This is the name of the data source name (DSN) you are configuring.
Description*	This field provides a description of the DSN to help distinguish it from others.
Database type	This indicates for which type of database you are configuring this DSN (Oracle).
Driver*	On the **New Data Source Name** page, this is the name of the ODBC driver installed for the type of database selected in the **Database type** box (Oracle). It is a nonconfigurable field. On the **Edit Data Source Name** page, this is the absolute path name of the ODBC driver specified for this DSN. It is a configurable field.
ServerName*	This is the TNS name as defined in the tnsnames.ora file by the Oracle database client utility.

LogonID	This is the username required for accessing the database. If the username is not provided when configuring a system DSN, every connection string using this DSN must include the username. **Important Security Note:** To prevent access to a database by unauthorized users in shared Web hosting environments, it is recommended that the username and password be provided in each connection string, rather than in the system DSN.
Password	This is the password required for accessing the database. If the password is not provided when configuring a system DSN, every connection string using the DSN must include the password. **Important Security Note:** To prevent access to a database by unauthorized users in shared Web hosting environments, it is recommended that the username and password be provided in each connection string, rather than in the system DSN.
EnableDescribeParam	When this option is enabled (the check box is selected), all **StoredProcedure** arguments are returned as string types. This parameter is enabled by default.
ProcedureRetResults	When this option is enabled (the check box is selected), Oracle returns record sets from a **StoredProcedure** call. This parameter is enabled by default.

***** Required parameters

Oracle Parameters (without Client)

The following table describes the Oracle 8i (8.1.6 and 8.1.7) and 9i database parameters (without client) available for configuring system DSNs, as they appear on the Sun Chili!Soft ASP Administration Console **New Data Source Name** and **Edit Data Source Name** pages. For more information, see "Configuring Data Source Names (DSNs)" in this chapter.

Parameter	Explanation
DSN*	This is the name of the data source name (DSN) you are configuring.
Description*	This field provides a description of the DSN to help distinguish it from others.
Database type	This indicates for which type of database you are configuring this DSN (Oracle).
Driver*	On the **New Data Source Name** page, this is the name of the ODBC driver installed for the type of database selected in the **Database type** box (Oracle). It is a nonconfigurable field.

	On the **Edit Data Source Name** page, this is the absolute path name of the ODBC driver specified for this DSN. It is a configurable field.
HostName	This is the computer on which the Oracle server resides. If your network supports named servers, you can specify a host name (such as `Oracleserver`). Otherwise, specify an IP address.
PortNumber	This is the port on which the database server is configured to listen. Ask your database administrator for this information.
SID	This is the Oracle System Identifier that refers to the instance of Oracle running on the server. You must provide this information when connecting to servers that support more than one instance of an Oracle database.
LogonID	This is the username required for accessing the database. If the username is not provided when configuring a system DSN, every connection string using this DSN must include the username. **Important Security Note:** To prevent access to a database by unauthorized users in shared Web hosting environments, it is recommended that the username and password be provided in each connection string, rather than in the system DSN.
Password	This is the password required for accessing the database. If the password is not provided when configuring a system DSN, every connection string using the DSN must include the password. **Important Security Note:** To prevent access to a database by unauthorized users in shared Web hosting environments, it is recommended that the username and password be provided in each connection string, rather than in the system DSN.
EnableDescribeParam	When this option is enabled (the check box is selected), all **StoredProcedure** arguments are returned as string types. This parameter is enabled by default.
ProcedureRetResults	When this option is enabled (the check box is selected), Oracle returns record sets from a **StoredProcedure** call. This parameter is enabled by default.
CatalogOptions	When this option is enabled (the check box is selected), the result column **REMARKS** for the catalog functions **SQLTables** and **SQLColumns**, and the result column **COLUMN_DEF** for the catalog function **SQLColumns**,

	will have meaning for Oracle. Enabling this option reduces the performance of your queries. This option is disabled by default, which returns **SQL_NULL_DATA** for the result columns **COLUMN_DEF** and **REMARKS**.
EnableStaticCursorsForLongData	When this option is enabled (the check box is selected), the driver supports long columns when using a static cursor. Enabling this option causes a performance penalty at the time of execution when reading long data. This option is disabled by default.
ApplicationUsingThreads	When this option is enabled (the check box is selected), the driver works with multi-threaded applications. When enabled, the driver is thread-safe. This option is enabled by default.

***** Required parameters

See also:

Configuring a Database in this chapter

Configuring Database Parameters in this chapter

PostgreSQL Parameters (Solaris and Linux only)

The following table describes the PostgreSQL 6.5.2 and 7.1.3 database parameters available for configuring system DSNs, as they appear on the Sun Chili!Soft ASP Administration Console **New Data Source Name** and **Edit Data Source Name** pages. For more information, see "Configuring Data Source Names (DSNs)" in this chapter.

Parameter	Explanation
Data Source Name*	This is the name of the data source name (DSN) you are configuring.
Database type*	This indicates for which type of database you are configuring this DSN (PostgreSQL).
Description	This field provides a description of the DSN to help distinguish it from others.
Driver*	On the **New Data Source Name** page, this is the name of the ODBC driver installed for the type of database selected in the **Database type** box (PostgreSQL). It is a nonconfigurable field. On the **Edit Data Source Name** page, this is the absolute path name of the ODBC driver specified for this DSN. It is a configurable field.
ServerName*	This is the IP address of the PostgreSQL database server. If this field is empty, the server is assumed to be running on the local computer.
Port*	This is the port on which the PostgreSQL database server is configured to listen. The default is 5432.

Database*	Because there can be multiple installations of a database running on one computer, each database is given its own name. This parameter indicates the name of the database for which you are configuring this DSN.
User	This is the username required for accessing the database. If the username is not provided when configuring a system DSN, every connection string using this DSN must include the username. **Important Security Note:** To prevent access to a database by unauthorized users in shared Web hosting environments, it is recommended that the username and password be provided in each connection string, rather than in the system DSN.
Password	This is the password required for accessing the database. If the password is not provided when configuring a system DSN, every connection string using the DSN must include the password. **Important Security Note:** To prevent access to a database by unauthorized users in shared Web hosting environments, it is recommended that the username and password be provided in each connection string, rather than in the system DSN.
ReadOnly*	When this option is enabled (the check box is selected), the database returns all record sets as read-only. This parameter is disabled by default.
UseCursorLib	When this option is enabled (the check box is selected), ODBC Manager cursor support overrides ODBC driver cursor support. Use this to enable scrollable cursors not supported by the driver. This parameter is enabled by default.

***** Required parameters

See also:

Configuring a Database in this chapter

Configuring Database Parameters in this chapter

SequeLink Parameters

You use SequeLink for connecting to Microsoft Access and Microsoft SQL Server 6.5 databases running on Windows-based computers. The following table describes the SequeLink database parameters available for configuring system DSNs, as they appear on the Sun Chili!Soft ASP Administration Console **New Data Source Name** and **Edit Data Source Name** pages.

Parameter	Explanation
Data Source Name*	This is the name of the data source name (DSN) you are configuring. It must match the entry for **SQLnkDSN** (below), which is the DSN created with the ./setsqllnk utility. For more information, see "Configuring SequeLink" in this chapter.
Database type	On the **New Data Source Name** page, select **SequeLink 4.51a** from the list to configure a DSN for a Microsoft Access or Microsoft SQL Server 6.5 database. On the **Edit Data Source Name** page, **SequeLink 4.51a** appears in this field.
Description*	This field provides a description of the DSN to help distinguish it from others.
Driver*	On the **New Data Source Name** page, this is the name of the database driver configured for this DSN (SequeLink). It is a nonconfigurable field. On the **Edit Data Source Name** page, this is the absolute path name of the database driver specified for this DSN. It is a configurable field.
SQLnkDSN*	This is the name of the DSN configured by running the ./setsqllnk utility. For more information, see "Configuring SequeLink" in this chapter.
Database*	For Microsoft Access databases, this is the absolute path name of the Access MDB file on the Windows-based server. For Microsoft SQL Server 6.5 databases, this is the database name of the SQL Server database.
LogonID	This is the username required for accessing the database. If the username is not provided when configuring a system DSN, every connection string using this DSN must include the username. **Important Security Note:** To prevent access to a database by unauthorized users in shared Web hosting environments, it is recommended that the username and password be provided in each connection string, rather than in the system DSN.
PreFetchRows	This is an advanced feature. Do not change this setting without first contacting Sun Chili!Soft Customer Support. The default is **30**.
Password	This is the password required for accessing the database. If the password is not provided when configuring a system DSN, every connection string using the DSN must include the password.

	Important Security Note: To prevent access to a database by unauthorized users in shared Web hosting environments, it is recommended that the username and password be provided in each connection string, rather than in the system DSN.
EnableWarnings	This is an advanced feature. Do not change this setting without first contacting Sun Chili!Soft Customer Support. This parameter is enabled by default (the check box is selected).
UidPwdMapping	Microsoft SQL Server requires two sets of usernames and passwords, one for accessing the host server, and one for accessing the database. Only one pair can be passed via the connection string (or the LoginID/password configured here). The other must be configured into the SqlnkDSN, as described in "Configuring SequeLink" in this chapter. When **UidPwdMapping** is enabled (the check box is selected), the LoginID represents the host. When **UidPwdMapping** is disabled (the check box is not selected) the LoginID represents the database. This parameter is disabled by default.
AllowBatchStatements	This is an advanced feature. Do not change this setting without first contacting Sun Chili!Soft Customer Support. This parameter is disabled by default (the check box is not selected).
EnableScrollableCursors	This option enables the use of cursor types other than "ForwardOnly." This parameter is enabled by default (the check box is selected).
DataDictionary	This is an advanced feature. Do not change this setting without first contacting Sun Chili!Soft Customer Support.

***** Required parameter

See also:

Configuring SequeLink in this chapter

Configuring a Database in this chapter

Configuring Database Parameters in this chapter

Sybase Parameters

The following table describes the Sybase 11.9.2 or 12.5 database parameters available for configuring system DSNs, as they appear on the Sun Chili!Soft ASP Administration Console **New Data Source Name** and **Edit Data Source Name** pages. For more information, see "Configuring Data Source Names (DSNs)" in this chapter.

Parameter	Explanation
DSN*	This is the name of the data source name (DSN) you are configuring.
Description	This field provides a description of the DSN to help distinguish it from others.
Database type*	This indicates for which type of database you are configuring this DSN (Sybase).
Driver	On the **New Data Source Name** page, this is the name of the ODBC driver installed for the type of database selected in the **Database type** box (Sybase). It is a nonconfigurable field. On the **Edit Data Source Name** page, this is the absolute path name of the ODBC driver specified for this DSN. It is a configurable field.
Server*	This is the IP address of the Sybase database server. If this field is empty, the server is assumed to be running on the local computer.
Port*	This is the port of the Sybase database server.
Database*	Because there can be multiple installations of a database running on one computer, each database is given its own name. This parameter indicates the name of the database for which you are configuring this DSN.
LogonID*	This is the username required for accessing the database. If the username is not provided when configuring a system DSN, every connection string using this DSN must include the username. **Important Security Note:** To prevent access to a database by unauthorized users in shared Web hosting environments, it is recommended that the username and password be provided in each connection string, rather than in the system DSN.
Password*	This is the password required for accessing the database. If the password is not provided when configuring a system DSN, every connection string using the DSN must include the password. **Important Security Note:** To prevent access to a database by unauthorized users in shared Web hosting environments, it is recommended that the username and password be provided in each connection string, rather than in the system DSN.

*Required parameters

See also:

Configuring a Database in this chapter

Configuring Database Parameters in this chapter

Text Parameters

The following table describes the Text parameters available for configuring system DSNs, as they appear on the Sun Chili!Soft ASP Administration Console **New Data Source Name** and **Edit Data Source Name** pages. For more information, see "Configuring Data Source Names (DSNs)" in this chapter.

Parameter	Explanation
DSN*	This is the name of the data source name (DSN) you are configuring.
Description	This field provides a description of the DSN to help distinguish it from others.
Database type*	This indicates for which type of database you are configuring this DSN.
Driver*	This is the installed OBDC driver specified for this DSN.
Database*	This is the directory in which the text files are stored. If left empty, the current working directory is used.
IntlSort	This field determines the order in which records are retrieved when you issue a **Select** statement with an **Order By** clause. When set to **0** (the default), ASCII sort order is used. Items are sorted alphabetically, with uppercase letters preceding lowercase letters (for example, "A, b, C" would be sorted as "A, C, b"). When set to **1**, international sort order is used, as defined by your operating system. The order is always alphabetic, regardless of case.

* Required parameters

See also:

Configuring a Database in this chapter

Configuring Database Parameters in this chapter

Configuring ActiveX Data Objects (ADO) Connections

ActiveX Data Objects (ADO) is the Microsoft standard for database access. Sun Chili!Soft ASP provides an ADO control, which you can configure by using the Sun Chili!Soft ASP Administration Console. For more information about ADO, see "Enabling Database Connections on the Server" in "Chapter 2: Installing and Configuring Sun Chili!Soft ASP," and "ADO Component Reference" in "Chapter 5: Developer's Reference."

This section describes the following configuration options for the ADO control:

- Setting the ADO Connection Pool Size
- Enabling and Disabling ADO Logging

Setting the ADO Connection Pool Size

Sun Chili!Soft ASP supports database connection pooling, which improves the performance of applications that rely heavily on database operations. With connection pooling, rather than opening and closing a database connection for each individual request, Sun Chili!Soft ASP uses a connection that is already open.

Sun Chili!Soft ASP uses an ADO control to provide database connectivity. You set the ADO connection pool size parameter by using the Sun Chili!Soft ASP Administration Console. The default ADO connection pool size is 25, which you can either increase or decrease according to your requirements. Setting this parameter to 0 (zero) disables connection pooling.

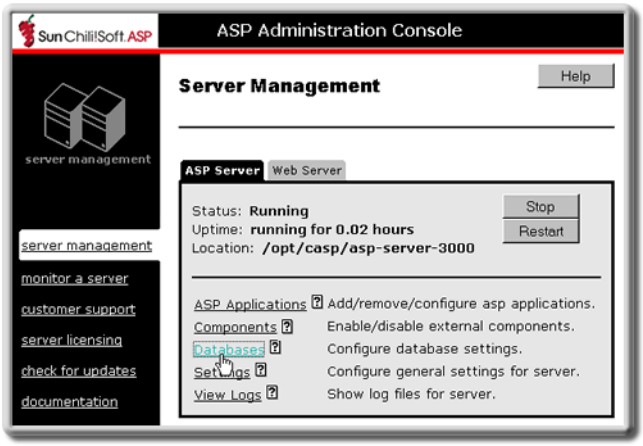
To set the ADO connection pool size

1. Open the Administration Console by using the following URL:

   http://[HOSTNAME]:[PORT]

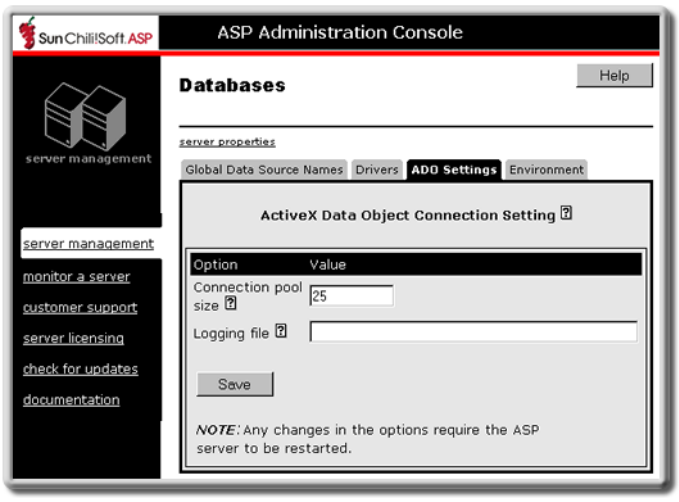
   where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

2. On the **ASP Server** tab of the **Server Management** page (the first page to display when you open the Administration Console), click **Databases**.



3. Click the **ADO Settings** tab.

   The **ActiveX Data Object Connection Setting** page displays.



4. In the **Connection pool size** box, type the number of connections you want to pool.
5. Click **Save**, and then click **server management** in the left navigation pane.
6. Restart the ASP Server by clicking **Restart**.

Note

Restarting the ASP Server resets all **Session** and **Application** variables.

See also:

Pooling Database Connections in this chapter

Enabling ADO Logging

Sun Chili!Soft ASP uses an ADO control to provide database connectivity. Logging for ADO is enabled from the Sun Chili!Soft ASP Administration Console by providing an absolute path name for the log file. When you do this, Sun Chili!Soft ASP creates the log file in the directory you specify and begins logging to it. To disable logging, simply delete the path name of the log file.

ADO logging should be used only for diagnostic purposes, and should not be enabled when running Sun Chili!Soft ASP on a production server. ADO logging will not be functional if **Inherit user security** is set to **no**. For information about this setting, see "Setting the Security Mode" in this chapter.

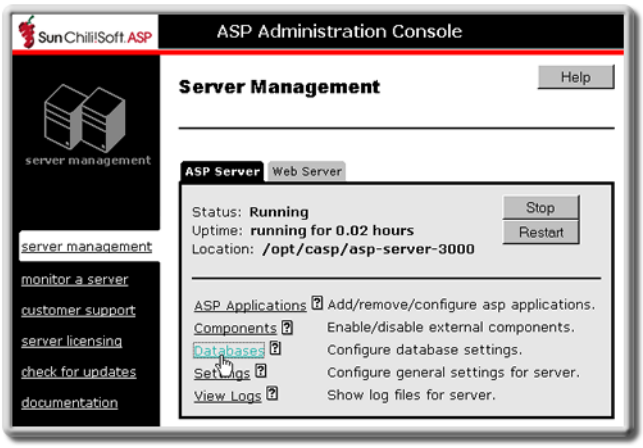
To enable ADO logging

1. Open the Administration Console by using the following URL:

   http://[HOSTNAME]:[PORT]

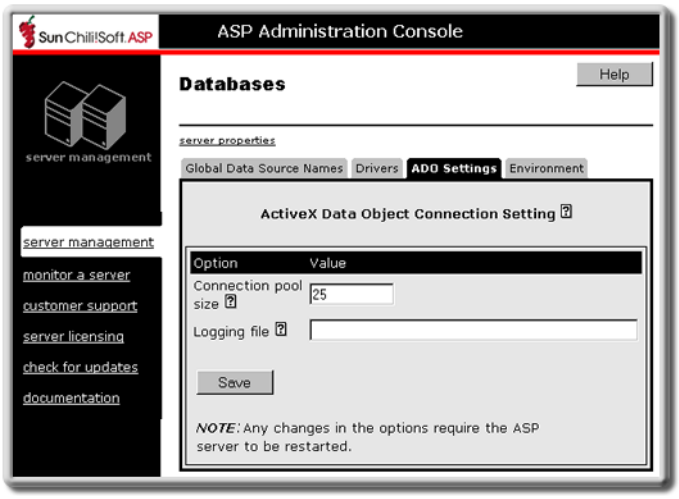
   where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

2. On the **ASP Server** tab of the **Server Management** page (the first page to display when you open the Administration Console), click **Databases**.



3. Click the **ADO Settings** tab.

   The **ActiveX Data Object Connection Setting** page displays.



4. In the **Logging file** box, type the absolute path name of the log file. This includes the path to the directory containing the file and the name of the log file. You cannot use the name of a file that already exists in the directory.
5. Click **Save**, and then click **server management**.
6. Restart the ASP Server by clicking **Restart**.

Note

Restarting the ASP Server resets all **Session** and **Application** variables.

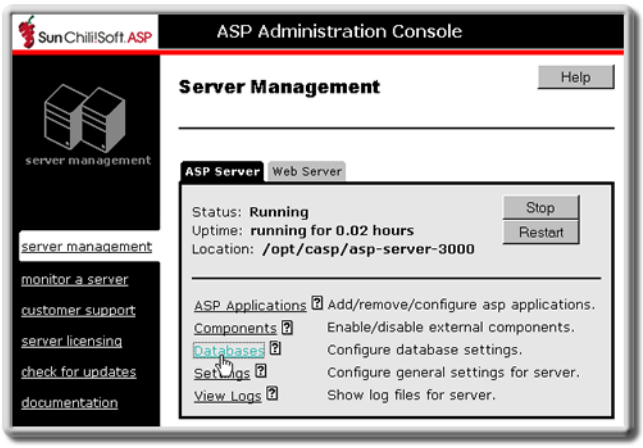
To disable ADO logging

1. Open the Administration Console by using the following URL:

http://[HOSTNAME]:[PORT]

where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

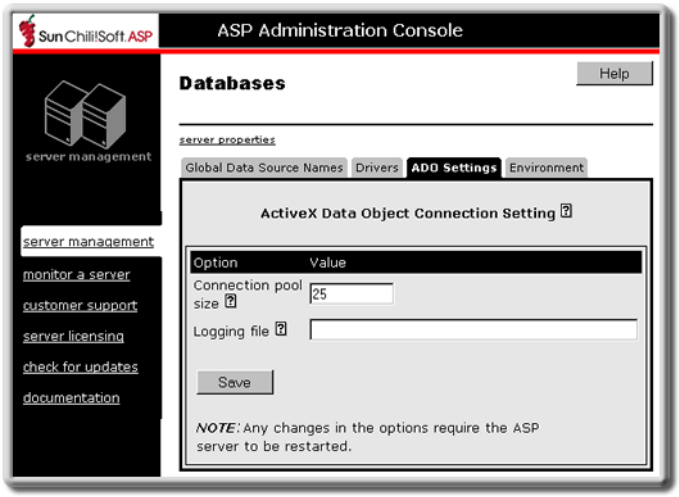
2. On the **ASP Server** tab of the **Server Management** page (the first page to display when you open the Administration Console), click **Databases**.



The **Databases** page displays.

3. Click the **ADO Settings** tab.

The **ActiveX Data Object Connection Setting** tab displays.



4. Delete the text in the **Logging file** box.
5. Click **Save**, and then click **server management**.
6. Restart the ASP Server by clicking **Restart**.

Note

Restarting the ASP Server resets all **Session** and **Application** variables.

See also:

Configuring ActiveX Data Objects (ADO) Connections in this chapter

Running Sun Chili!Soft ASP in a Shared Web Hosting Environment

Sun Chili!Soft ASP supports the scenario in which users share physical hardware and a Web server, such as with an Internet Service Provider (ISP) or Internet Presence Provider (IPP). In a shared Web hosting environment, a single Web server installation answers requests for multiple domain names by using virtual hosts. This section provides information about running Sun Chili!Soft ASP in a shared Web hosting environment.

In this section:

- Creating Database Connections in a Shared Environment
- Defining Applications in a Shared Environment
- Using the User Configuration File
- Using the FrontPage Services File in a Shared Environment

The following topic in this chapter describes security information you should be aware of when configuring a shared Web hosting environment:

- Securing the Server

The following section in this chapter describes advanced options for administering Sun Chili!Soft ASP, which can give you more flexibility when configuring a shared hosting environment:

- Advanced Administration Options

Creating Database Connections in a Shared Environment

With Sun Chili!Soft ASP, ASP developers can specify the connection information for a database by using either system DSNs, file DSNs, or DSN-less connection strings. The appropriate method to use depends on user preferences and the environment in which Sun Chili!Soft ASP is running.

In enterprises and other dedicated hosting environments, it is recommended that ASP developers use system DSNs. The system administrator uses the Sun Chili!Soft ASP Administration Console to create system DSNs, which then can be referenced from within an ASP application for initializing a database connection. For more information, see "Configuring Data Source Names (DSNs)" in this chapter.

However, in a shared Web hosting environment, such as with an Internet Service Provider (ISP), system DSNs pose two problems:

- System DSNs can be a security risk. System DSNs can include the username and password required for accessing the database, making the data source accessible from any ASP page on the server.

- Creating DSNs for each customer can create a significant administrative burden for the Web hosting provider. Because Web developers can create them, and database access is restricted to the specific ASP application using the connection, file DSNs and DSN-less connection strings are often more appropriate in a Web hosting environment.

See also:

Configuring a Database in this chapter

Enabling Database Connections on the Server in "Chapter 2: Installing and Configuring Sun Chili!Soft ASP"

Using DSN-less Connection Strings" in "Chapter 4: Building a Sun Chili!Soft ASP Application"

Defining Applications in a Shared Environment

For the ASP Server to process an ASP application, the set of directories and files comprising the application must be defined as an ASP application. In dedicated Web hosting environments, you define an ASP application by "adding" it to the ASP Server, as described in "Configuring ASP Applications" in this chapter.

However, in a shared Web hosting environment in which you are using virtual hosts, such as with an Internet Service Provider (ISP) or Internet Presence Provider (IPP), you do not "add" applications in this manner. Instead, the top-level, or root, directory of each virtual host defined on your Web server is automatically defined as an ASP application. No other steps are necessary to enable ASP processing for the application.

There might be some situations, however, in which you want to enable or disable ASP processing for a particular virtual host. You can do this by using the Sun Chili!Soft ASP Administration Console, as described in "Enabling ASP for a Virtual Host" in this chapter.

FrontPage users can also define an application as described in "Using the FrontPage Services File in a Shared Environment" in this chapter.

Using the User Configuration File

In a shared Web hosting environment, rather than requiring the system administrator to define each ASP application (as described in "Adding an ASP Application" in this chapter), you can enable ASP developers to define their own ASP applications in a User Configuration file. To do this, the system administrator must first edit the Sun Chili!Soft ASP configuration file, casp.cnfg, so that the ASP Server recognizes applications that are defined in the User Configuration file. Then ASP developers can create the file and define their ASP applications within it.

To enable developers to define their own ASP applications, take the following steps. First, in the `[applications]` section of the Sun Chili!Soft ASP configuration file, casp.cnfg, specify the path name of the User Configuration file (.aspconf) that defines the ASP applications.

```
[applications]
config_name=.aspconf
```

When you do this, the ASP Server looks for this file in the document root of the Web server and each virtual host. For more information about editing casp.cnfg, see "Editing the Chili!Soft Configuration File" in this chapter.

Next, create a User Configuration file. It should be a plain text file named .aspconf. Within this file, specify the ASP application name to define as follows:

```
[applications]

/[appname]
```

where `[appname]` is the ASP application name. The ASP application name must be the same as the name of the ASP application root directory, which is contained in the document root of the virtual host.

Any applications defined in the User Configuration file are dynamically recognized, without requiring the ASP Server to be restarted.

There are two limitations on applications defined in the User Configuration file. First, the application directory containing the global.asa file must be directly below the top-level directory of the Web server or virtual host document root. Second, if the User Configuration file appears in the document root of a virtual host, then the ASP applications are applied only to that virtual host, and not to others.

See also:

Configuring ASP Applications in this chapter

Using the FrontPage Services File in a Shared Environment

In a shared Web hosting environment, you can enable developers to define new ASP applications by using FrontPage. You can use FrontPage to create new global.asa files and ASP applications. FrontPage stores the definitions of these new applications in the FrontPage services.cnf file in the /_vti_pvt subdirectory.

Sun Chili!Soft ASP automatically looks for the services.cnf file in the /_vti_pvt subdirectory, and treats the entries it finds in this file as ASP applications. Any applications defined in the services.cnf file are dynamically recognized by Sun Chili!Soft ASP, and do not require the ASP Server to be restarted. Sun Chili!Soft ASP looks for this filename in the document root directory of the Web server (and each virtual host).

Entries in the services.cnf file use the following format:

```
/[appname] = "/path/to/app/home/directory"
```

If the services.cnf file and the /_vti_pvt subdirectory appear in the document root directory of a virtual host, then the ASP applications are applied only to that virtual host, and not to others.

There are two limitations on applications defined in the services.cnf file. First, the files in the application must be located within the document root directory of the Web server (or virtual host). Second, the directory containing the global.asa file cannot be below the top-level directory

of the Web server (or virtual host) document root. For more information about ASP applications and the global.asa file, see "Configuring ASP Applications" in this chapter.

See also:

Adding an ASP Application in this chapter

Defining Applications in a Shared Evironment in this chapter

Optimizing Server Performance

Sun Chili!Soft ASP has many features that enhance its scalability and performance. This section discusses those features, and includes the following topics:

- Enabling Scripts Buffering
- Changing the Session Timeout Value
- Changing the Script Timeout Value
- Enabling Script Caching
- Configuring Multi-threading
- Precompiling ASP Pages
- Pooling Database Connections
- Load Balancing

See also:

Enabling ASP Error Logging in this chapter

Monitoring the ASP Server in this chapter

Viewing Information About the ASP Server in this chapter

Viewing Log Files in this chapter

Enabling Scripts Buffering

Sun Chili!Soft ASP enables you to buffer ASP scripts to improve server performance. When scripts buffering is enabled, the ASP Server waits until the entire ASP page is processed before returning the results to the browser. When scripts buffering is disabled, the ASP Server returns the HTML output for an ASP page to the browser incrementally, as soon as it is processed. For a production server, it is best to enable scripts buffering. During development, however, you might want to disable scripts buffering so you can more easily debug problems with your ASP pages.

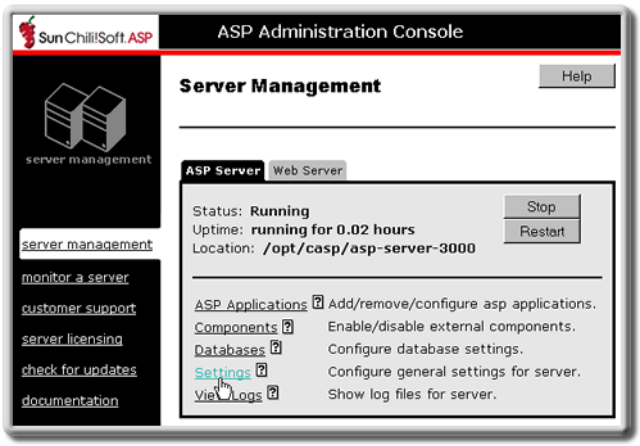
To enable or disable scripts buffering

1. Open the Administration Console by using the following URL:

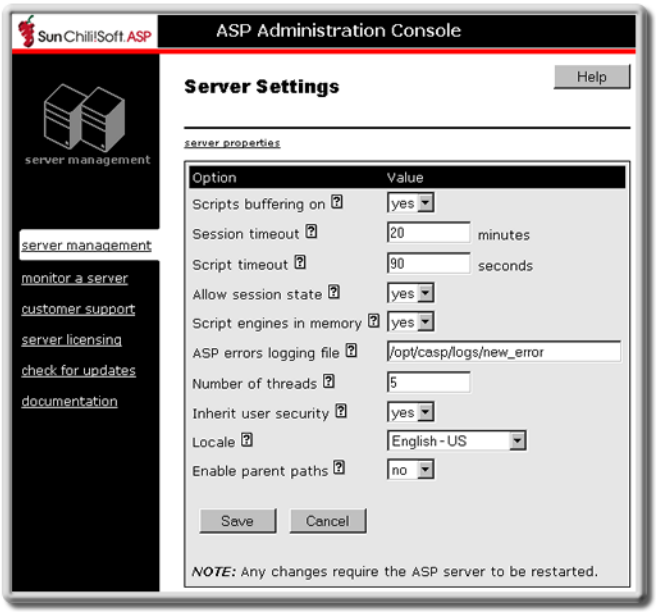
   http://[HOSTNAME]:[PORT]

where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

2. On the **ASP** tab of the **Server Management** page (the first page that displays when you open the Administration Console), click **Settings**.



The **Server Settings** page displays.



3. In the **Scripts buffering on** drop-down list, select **yes** to enable scripts buffering or **no** to disable it.

4. Click **Save**, and then restart the ASP Server by clicking **Restart**.

Note

Restarting the ASP Server resets all **Session** and **Application** variables.

See also:

Optimizing Server Performance in this chapter

Changing the Session Timeout Value

You can specify the number of minutes that the ASP Server maintains a user's session information since the last page request. When the user does not submit a request for the specified length of time, the server cancels the session and discards its stored information. Enabling the ASP Server to discard user information frees up its resources for another session.

By default, the session timeout value is 20 minutes. To change this value, use the following procedure.

Note

A value specified for **SessionTimeout** in a script overrides this setting.

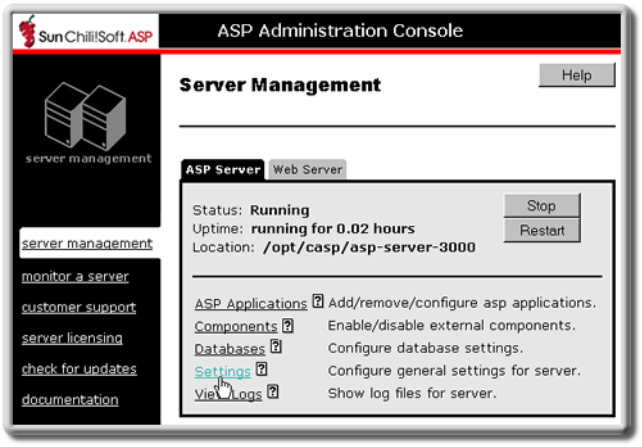
To change the session timeout value

1. Open the Administration Console by using the following URL:

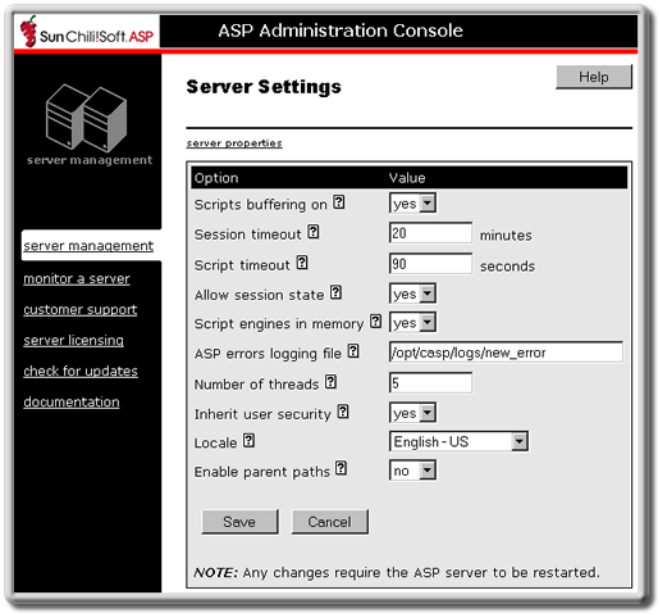
   http://[HOSTNAME]:[PORT]

   where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

2. On the **ASP** tab of the **Server Management** page (the first page that displays when you open the Administration Console), click **Settings**.



The **Server Settings** page displays.



3. In the **Session timeout** box, type the number of minutes of inactivity after which a user session times out.
4. Click **Save**, and then restart the ASP Server by clicking **Restart**.

Note

Restarting the ASP Server resets all **Session** and **Application** variables.

See also:

Optimizing Server Performance in this chapter

Changing the Script Timeout Value

You can specify the number of seconds that the ASP Server waits for an ASP page to finish processing before canceling the page request. Setting a script timeout prevents a malfunctioning ASP page from indefinitely engaging server resources. Enabling the ASP Server to cancel a page request frees up its resources for another session.

By default, the session timeout value is 90 seconds. To change this value, use the following procedure.

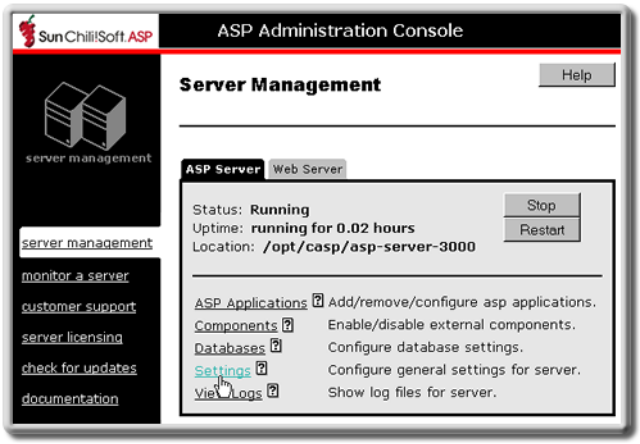
To change the script timeout value

1. Open the Administration Console by using the following URL:

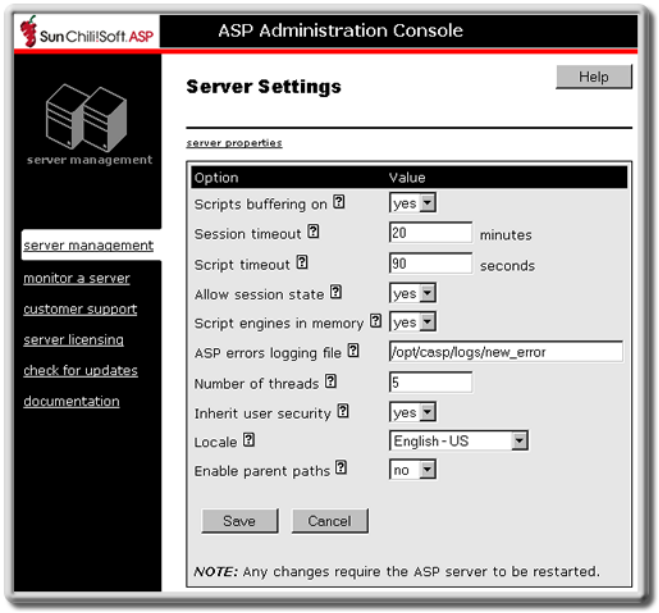
   http://[HOSTNAME]:[PORT]

   where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

2. On the **ASP** tab of the **Server Management** page (the first page that displays when you open the Administration Console), click **Settings**.



The **Server Settings** page displays.



3. In the **Script timeout** box, type the number of seconds after which a script should time out.
4. Click **Save**, and then restart the ASP Server by clicking **Restart**.

Note

Restarting the ASP Server resets all **Session** and **Application** variables.

See also:

Optimizing Server Performance in this chapter

Enabling Script Caching

You can enable the ASP Server to cache ASP scripts in memory so that it can serve ASP pages more quickly. Script caching is enabled by default. To change this value, use the following procedure.

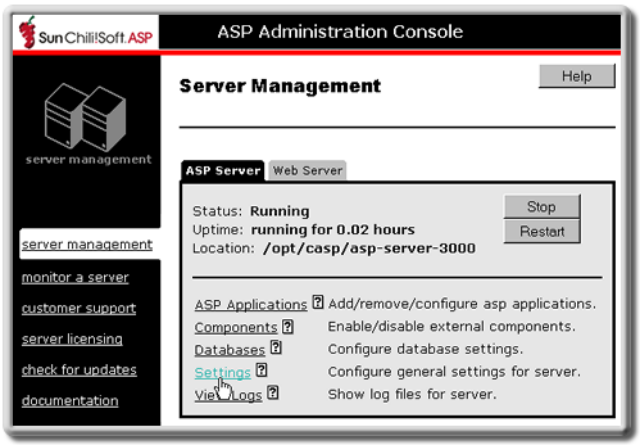
To enable or disable script caching

1. Open the Administration Console by using the following URL:

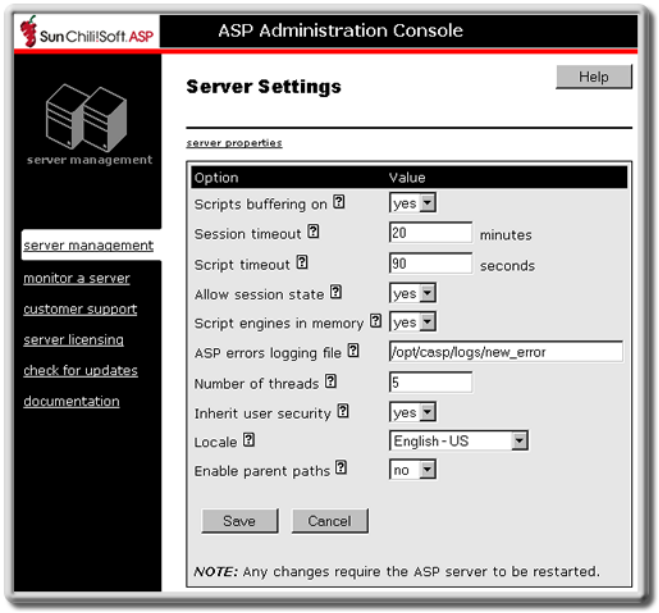
   http://[HOSTNAME]:[PORT]/

   where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

2. On the **ASP** tab of the **Server Management** page (the first page that displays when you open the Administration Console), click **Settings**.



The **Server Settings** page displays.



3. In the **Script engines in memory** drop-down list, select **yes** or **no** as desired.
4. Click **Save**, and then restart the ASP Server by clicking **Restart**.

Note

Restarting the ASP Server resets all **Session** and **Application** variables.

See also:

Optimizing Server Performance in this chapter

Configuring Multi-threading

The number of threads handled by the Sun Chili!Soft ASP Server at a time is set to 10 by default. If you have many ASP pages that include blocking operations (database access, for example) it is a good idea to increase this number. Keep in mind, however, that doing so creates more system overhead. A maximum number of 20 threads is recommended.

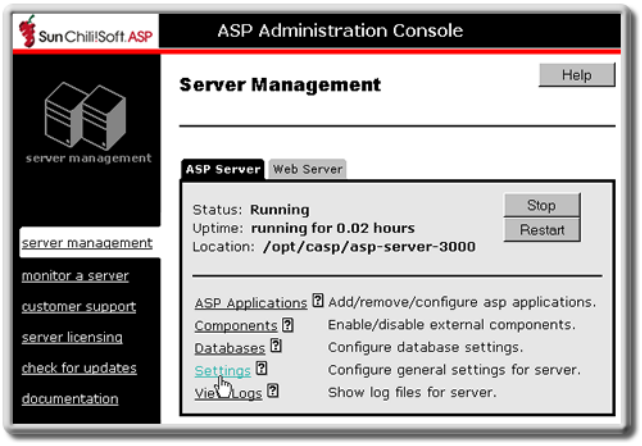
To configure Sun Chili!Soft ASP to use a specific number of threads

1. Open the Administration Console by using the following URL:

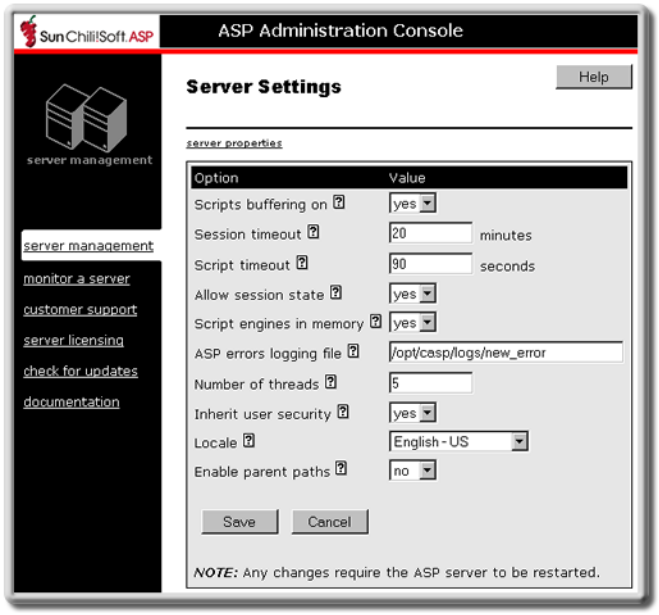
   http://[HOSTNAME]:[PORT]

   where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

2. On the **ASP** tab of the **Server Management** page (the first page that displays when you open the Administration Console), click **Settings**.



   The **Server Settings** page displays.



3. In the **Number of threads** box, type the maximum number of threads you want to have running at once. This number is **10** by default.
4. Click **Save**, and then click **Yes** to restart the Web server and ASP Server.

Note

Restarting the ASP Server resets all **Session** and **Application** variables.

Precompiling ASP Pages

The Sun Chili!Soft ASP Server automatically precompiles ASP pages to improve server performance. When the ASP Server receives a page request, it compiles the page into bytecode that can be more quickly processed in response to subsequent requests and saves the bytecode.

Pooling Database Connections

In terms of server resources, accessing a database is one of the most expensive operations of a Web application. Typically, for each request, the Web application must open a connection to the database, retrieve the data, and then close the connection. Repeatedly opening and closing the database adversely impacts server performance.

To reduce this impact on server performance, you can configure the Sun Chili!Soft ASP Server to share open database connections among multiple users who are accessing the Web application. This is called database connection pooling. With connection pooling, the ASP Server uses a connection that is already open, rather than opening and closing a database connection for each individual request. Database connection pooling dramatically improves the performance of applications that rely heavily on database operations.

To configure database connection pooling, use the procedure in "Setting the ADO Connection Pool Size" in this chapter.

Load Balancing

Sun Chili!Soft ASP supports various models for horizontal scalability and load balancing, including both software- and hardware-based solutions.

The classic model for providing horizontal scalability is to add additional servers to an overall "farm" of servers. The addition of user sessions, however, adds an element of complexity to the horizontal scalability picture. For ASP to maintain session information for a specific user, the user's requests must consistently be routed back to the same machine with which the initial session was created. This is called "session-aware load balancing," and can be done using either software or hardware solutions.

Sun Chili!Soft ASP supports both hardware- and software-based session-aware load balancing solutions. Software options are based primarily on round-robin DNS and clustering software, while hardware solutions include the use of "intelligent routers" (also referred to as "sticky sessions"). Intelligent routers are capable of routing a user's request back to the same machine with which the initial session was created.

Advanced Administration Options

This section describes options that advanced users may decide to use to configure the Sun Chili!Soft ASP Server from the command line, or by directly editing configuration files.

Caution

> Take great care when making the changes described in this section. Changes you make could require a complete reinstall of Sun Chili!Soft ASP, and could void your eligibility for customer support. You should back up your data before making any changes.
>
> For UNIX and Linux systems, most of the configuration settings described in this section are easily accessed from the Sun Chili!Soft ASP Administration Console. It is strongly recommended that you use the Administration Console whenever possible. For more information, see "Using the Administration Console" in this chapter.

In this section:

- Editing the Windows Registry
- Editing the Sun Chili!Soft ASP Configuration File
- Using the caspctrl Script
- Defining Applications on UNIX
- Relocating the System Files for a Shared Installation

Editing the Windows Registry

Sun Chili!Soft ASP for Windows stores some configuration information in the system registry. This topic describes the registry settings Sun Chili!Soft ASP uses. You can use regedit32 to edit these settings; regedit32 is installed with the operating system.

Caution

Take great care when making the changes described in this section. Changes you make could require a complete reinstall of Sun Chili!Soft ASP, and could void your eligibility for customer support. Be sure to back up your data before making any changes.

Key	Default Value	Description
AllowOutOfProcCmpnts	0 (False)	Controls whether Sun Chili!Soft ASP allows Active Server Components that do not run in the Sun Chili!Soft ASP process space. Out-of-process components run more slowly than in-process components, but they are safer because an individual component cannot bring down the ASP Server.
AllowSessionState	1 (True)	Controls whether the ASP Server maintains session state. If **AllowSessionState** is **False**, the **Session** object cannot be used.
BufferingOn	1 (True)	Controls whether the ASP Server processes the entire ASP page before returning HTML (BufferingOn = True), or whether it returns the HTML generated from an ASP page as the page is processed (BufferingOn = False). BufferingOn provides slightly better performance when set to **True**.
DefaultError	See Description	This value controls the message returned when the ASP Server encounters a run-time error and cannot process a page. It appears if **ShowDefaultError = True**. The default error message is: "An error occurred on the server when processing the URL. Please contact the system administrator."
DefaultLanguage	VBScript	This setting determines the language that the ASP Server assumes is used in ASP pages. The other option is JScript. This setting can be overridden in individual pages with an `@LANGUAGE` directive or `<SCRIPT>` block.
DefaultScriptLanguage	VBScript	This registry key is not active. See DefaultLanguage.
Enabled	1 (True)	Controls whether the ASP Server processes ASP pages. If **False**, when a user requests an ASP page, the ASP Server returns the message "Sun Chili!Soft ASP has been disabled and cannot process your request."

LogDirectory	See description	Default value = "c:\WINNT\System32\chiliasp" This value controls the directory to which the ASP Server writes the log file if **LogToFile** is **True**.
LogErrors	0 (False)	Determines if ASP Server errors should be written to the Sun Chili!Soft ASP log file.
LogToFile	0 (False)	This registry entry is set internally by the Sun Chili!Soft ASP engine to control logging of debug information. *Do not modify this setting.*
LogRequestErrors	0 (False)	Creates a file named "errors" in the directory specified by LogDirectory.
MaxThreads	10	This value controls the maximum number of threads per CPU that the Sun Chili!Soft ASP engine uses to process requests.
NumInitialThreads	2	The number of threads to start when the ASP Server starts. *This feature is not currently implemented. Use MaxThreads instead.*
Reset	00000000	This is a Sun Chili!Soft ASP internal setting. *Do not modify this setting*.
Running	00000001	This registry entry is set internally by the Sun Chili!Soft ASP engine to indicate whether Sun Chili!Soft ASP is running. *Do not modify this setting*.
ScriptEngineCacheMax	ffffffff	This value controls the maximum number of script engines that Sun Chili!Soft ASP caches for servicing ASP page requests. *This feature is not completely implemented. The default setting turns caching on; any other setting turns caching off.*
ScriptTimeout	90 Seconds	This is the amount of time the ASP Server waits for an individual ASP page to finish processing before canceling the request. The ScriptTimeout value can be increased in a script, but this value sets the minimum.
SessionTimeout	20 minutes	This value controls how long the ASP Server maintains Session values for a user without receiving a page request. If the user is not heard from in this amount of time, the session is canceled and its values are discarded. The SessionTimeout value can be increased in a script, but this value is the minimum.
ShowDefaultError	0 (False)	This value controls ASP Server response to run-time errors. If **True**, the ASP Server returns a message in DefaultError when a run-time error occurs.
StartConnectionPool	1 (True)	If **True**, enables connection pooling when connecting to a

database.

Editing the Sun Chili!Soft ASP Configuration File

UNIX and Linux versions of Sun Chili!Soft ASP include a configuration file, casp.cnfg, in which you can change Sun Chili!Soft ASP settings. This topic describes the settings and their parameters.

Caution

> Take great care when making the changes described in this section. Changes you make could require a complete reinstall of Sun Chili!Soft ASP, and could void your eligibility for customer support. You should back up your data before making any changes.
>
> Most of the configuration settings described in this section are easily accessed from the Sun Chili!Soft ASP Administration Console. It is strongly recommended that you use the Administration Console whenever possible, as described in "Using the Administration Console" in this chapter.

You can find the casp.cnfg file in the following location:

/[C-ASP_INSTALL_DIR]/asp-server-[PORT]

where [C-ASP_INSTALL_DIR] is the path name of the Sun Chili!Soft ASP installation directory, and [PORT] is the ASP Server port number (resembles 3000).

You can open casp.cnfg in any text editor and make the changes you want. For the changes to take effect, you must restart the ASP Server, as described in "Stopping and Restarting the ASP Server" in this chapter.

The casp.cnfg file is divided into sections by keywords. The following sections describe the keywords and parameters for each section.

[machines]

The `[machines]` keyword defines the computers that are running the Sun Chili!Soft ASP Server. The parameters specified in this section affect all Sun Chili!Soft ASP Servers.

Parameters

`count`

The number of computers running the ASP Server.

`machine1 … machineN`

The IP address of each computer running the ASP Server. The number of entries should be the same as the number of computers running an ASP Server.

`portnumber`

The base IP port to which the ASP Server control process listens. In multi-threading mode, the ASP Server uses two ports.

logfile

Defines the name and location of the ASP Server status log file.

mtengine (1)

Controls multi-threading in the ASP server. When `mtengine` is set to 1, the ASP Server runs one process with multiple threads to serve requests.

disablerestart

This setting is useful for Sun Chili!Soft ASP diagnostics. If set to 1, the Sun Chili!Soft ASP parent process does not automatically re-spawn Sun Chili!Soft ASP child processes that fail.

hashobj_pid

(Optional) This setting enables you to specify the name and location of the process ID (PID) file for the Sun Chili!Soft ASP hash object.

[default machine]

The `[default machine]` keyword defines a section containing parameters that control the operation of the ASP Server on each computer.

Parameters

license

The absolute path name of the directory containing the Sun Chili!Soft ASP license file.

caspd_pid

(Optional) The name and location of the process ID (PID) file for the Sun Chili!Soft ASP daemon.

maxprocesses (1 to 20)

The maximum number of ASP Server threads that are used to process pending ASP requests. The number specified can be between 1 and 20. I/O-heavy scripts run better with more processes.

inherit_user (1/0)

This setting enables you to specify the security mode under which the ASP Server runs and can have a serious impact on the security of your server. In particular, if you are running iPlanet Web Server or Zeus Web Server, be sure to read the following "Important Security Information" note.

The ASP Server can run with the permissions of the user defined for the Apache Web Server or virtual host, with the permissions of a user or group defined in the casp.cnfg file, or with root permissions. You can specify the mode as follows:

- **Inherit User Security mode.** This mode, the default, is available only on Sun Chili!Soft ASP running with Apache Web Server. When `inherit_user=1`, the ASP Server runs with ("inherits") the permissions of the user defined for the Apache Web Server or virtual

host as defined in the Apache configuration file. This is the case even if a different user or group is specified in the `[default machine]` section of the casp.cnfg file (as discussed next).

- **Defined User Security mode**. This mode is available on Sun Chili!Soft ASP running with any supported Web server. In this mode, the ASP Server runs with the permissions of the user or group you specify. To run in this mode, set `inherit_user=0` and then specify the user or group in the `[default machine]` section of casp.cnfg, as described later.

Important Security Information

When `inherit_user=0` and no user or group is specified in the `[default machine]` section of casp.cnfg, the ASP Server runs as root. This can create a security risk for your server, so it is not recommended that you set `inherit_user=0` unless you also define a user or group for the ASP Server to run under.

iPlanet and Zeus Web servers do not support Inherit User Security mode, even when when **Inherit user security** is set to **yes** in the Administration Console. To protect the security of your server, when running Sun Chili!Soft ASP with these Web servers, specify a user or group in casp.cnfg for the ASP Server to run under.

For more information about the `inherit_user` setting, see "Setting the Security Mode" in this chapter.

`javasupport (yes/no)`

Yes enables Java support, which is required for Chili!Beans. **No**, the default, disables Java support. Because Java support can affect server performance, it is a good idea to enable it only when using Chili!Beans.

`Enablemonitoring (yes/no)`

Yes, the default, enables creation of performance counter log files, as follows:

/tmp/.casp[PORT]/chili-psm

/tmp/.casp[PORT]/.pm-chili-psm

/tmp/.pm-chili-psm

/tmp/chili-psm

These files are created with permissions that might not be appropriate in a shared Web hosting environment. **No** disables performance monitoring and the creation of these files.

`user`

(Optional) The username for the account under which the ASP Server runs. Make sure that this user has permission to open Sun Chili!Soft ASP configuration files such as casp.cnfg and odbc.ini. The user starting the ASP Server by using caspctrl must have root permissions. If this attribute is not present and `inherit_user=0`, the ASP Server runs under the account of the user that started the ASP Server.

`group`

(Optional) The group name for the account under which the ASP Server runs. Make sure that this group has permission to open Sun Chili!Soft ASP configuration files such as casp.cnfg and odbc.ini. The user starting the ASP Server using caspctrl must have greater permissions than this group. If this attribute is not present and `inherit_user=0`, the ASP Server runs under the account of the user that started the ASP Server.

[default application]

bufferingon (yes/no)

Yes enables script buffering.

sessiontimeout

Amount of time in seconds that the ASP Server waits for a new page request before canceling the session.

scripttimeout

Amount of time in seconds the ASP Server waits for an ASP page to finish processing before canceling the request.

allowsessionstate (yes/no)

Yes enables the use of the **Session** object in ASP scripts.

enableparentpaths (yes/no)

No, the default, limits file system access by the **FileSystemObject** to the application directory and subdirectories, and disables the use of **"../"** syntax. Yes enables access to the file system by the **FileSystemObject** outside the ASP application directory and the use of **"../"** syntax in **#include** and **Server.mapPath** statements.

defaultlanguage

Specifies the default script interpreter. This value can either be **vbscript** or **jscript**.

[ado]

connectionpoolsize

The number of ADO connections to pool (re-use) to improve server performance. The default is 25. "0" disables connection pooling.

logpath

Absolute path name of the ADO errors log file. Specifying the path name enables logging. You cannot use the name of a file that already exists in the directory.

maxlongfieldlength

Maximum long field length in bytes. By default this value is 65535. If the data you pass to a database exceeds this limit, the ODBC driver might crash. You can increase this value as needed.

[applications]

The `[applications]` keyword defines a section in which to specify information on how the ASP Server handles ASP applications. There are several ways to define an ASP application on the ASP Server. For more information, see "Configuring ASP Applications" in this chapter.

Parameters

`use_aliases`

If `use_aliases=yes`, then any virtual directory or alias defined in the Web server configuration file is treated as an ASP application. If `use_aliases=no`, then the virtual directories or aliases defined in the Web server configuration file are not treated as ASP applications by the ASP Server.

`/caspdoc`

Absolute path name of the directory containing the Sun Chili!Soft ASP product documentation.

`/caspadmin`

Absolute path name of the directory containing the admin files.

`/caspsamp`

Absolute path name of the directory containing the Sun Chili!Soft ASP samples.

`config_name`

(Optional) This parameter enables you to specify the name of the ASP User Configuration file. Any applications defined in this file are dynamically recognized by the ASP Server without requiring the ASP Server to be restarted. If `config_name=.aspconf`, for example, the ASP Server looks for this filename in the document root directory of the Web server. Entries in the config_name file should use the following format:

```
/[appname]
```

There are two limitations on applications defined in the ASP User Configuration file. First, the files in the application must be located within the document root of the Web server. Second, the directory containing the global.asa file must not be below the top-level directory of the Web server document root directory.

`/appname`

(Optional) To define an ASP application on the ASP Server, use the following format:

`/[appname] = "/[path_name]"` (the path name must be enclosed in double quotes).

where `[appname]` is the name specified for the application and [path_name] is the absolute path name of the directory containing the application files. If no applications are defined in the `[applications]` section, then the ASP Server treats the root directory of the Web server as the location of the "default" ASP application.

[virtual hosts]

(Optional) The `[virtual hosts]` keyword defines a section in which to configure the ASP Server to work with the virtual hosts feature of Apache Web Server. For more information, see "Defining Applications on UNIX," "Enabling ASP for a Virtual Host," and "Defining Applications in a Shared Environment" in this chapter.

Parameters

`allow_all`

(Optional) If `allow_all=no`, then ASP functionality is only enabled for the virtual host defined later in the `[virtual hosts]` section. If this attribute is omitted (or if `allow_all=yes`), ASP is enabled for all of the virtual hosts defined in the Web server configuration file.

`timeout`

(Optional) If a virtual host has had no ASP activity for the number of minutes specified in the timeout attribute, the ASP Server releases all of the cached ASP pages for that virtual host. The ASP Server does not time out a virtual host unless all of the sessions for that virtual host have timed out. If this setting is not configured, the default timeout is 60 minutes.

`hostID(s)`

(Optional) This setting applies only to Apache Web Server. It is a line-delimited list of hostnames that identify which virtual hosts are allowed to handle requests for ASP pages. The hostname(s) listed in this section should match the virtual hosts **ServerName** directive in the httpd.conf file of the Apache Web Server. This attribute becomes active if `allow_all=no`. If `allow_all=no` and no hostIDs are provided, ASP functionality is disabled for all virtual hosts.

Using the caspctrl Script

In addition to using the Administration Console, you can manage Sun Chili!Soft ASP by using the caspctrl script. This topic describes how to use the script and lists the options it provides.

Caution

> Take great care when making the changes described in this section. Changes you make could require a complete reinstall of Sun Chili!Soft ASP, and could void your eligibility for customer support. You should back up your data before making any changes.
>
> Most of the configuration settings described in this section are easily accessed from the Sun Chili!Soft ASP Administration Console. It is strongly recommended that you use the Administration Console whenever possible, as described in "Using the Administration Console" in this chapter.

To use the caspctrl script

- From the Sun Chili!Soft ASP installation directory, type `caspctrl` at the command prompt, followed by the options you want to use. The correct format is as follows:

```
caspctrl (-v|version) (-vc|verbose) [-
](startdaemon|stopdaemon|starteng|stopeng|startall|stopall|viewlog|c
learlog|status)
```

The options are as follows:

- **version:** reports the version of Sun Chili!Soft ASP
- **verbose:** enables the Sun Chili!Soft ASP engines to output status messages to stdout
- **startdaemon:** starts the Sun Chili!Soft ASP Server daemon
- **stopdaemon:** stops the Sun Chili!Soft ASP Server daemon (plus any running ASP engines)
- **starteng:** starts Sun Chili!Soft ASP engine(s) on all Sun Chili!Soft ASP computers in the configuration with running daemons
- **stopeng**: stops the Sun Chili!Soft ASP engine(s) on all computers running Sun Chili!Soft ASP in the configuration with running daemons
- **startall**: starts the Sun Chili!Soft ASP daemon and engine(s) on single-computer installations of Sun Chili!Soft ASP
- **stopall**: stops the Sun Chili!Soft ASP daemon and engine(s) on single-computer installations of Sun Chili!Soft ASP
- **viewlog:** enables you to view the Sun Chili!Soft ASP Server log
- **clearlog:** clears the Sun Chili!Soft ASP Server log
- **status:** reports the status of each Sun Chili!Soft ASP Server in the configuration

Defining Applications on UNIX

This topic describes the options that are available for defining Sun Chili!Soft ASP applications on UNIX-based systems. Some options might not be available on Cobalt platforms.

Caution

> Take great care when making the changes described in this section. Changes you make could require a complete reinstall of Sun Chili!Soft ASP, and could void your eligibility for customer support. You should back up your data before making any changes.
>
> Most of the configuration settings described in this section are easily accessed from the Sun Chili!Soft ASP Administration Console. It is strongly recommended that you use the Administration Console whenever possible, as described in "Using the Administration Console" in this chapter.

With Sun Chili!Soft ASP running on a UNIX system with any supported Web server, you can define an ASP application by using the following methods:

- Adding an entry to the `[applications]` section of the Sun Chili!Soft ASP configuration file, casp.cnfg. For more information, see "Editing the Sun Chili!Soft Configuration File" in this chapter.
- Adding an alias to the Web server configuration file (only if `use_aliases=yes` in the `[applications]` section of casp.cnfg).
- Adding an entry to the services.cnf file generated by FrontPage, located in the /_vti_pvt subdirectory of the Web server document root directory.

The ASP Server dynamically recognizes ASP applications that are defined in the Sun Chili!Soft ASP User Configuration file or the FrontPage services.cnf file. These applications must be defined by using the application name (for example, `"/appname"`). An application named `/customers` must correspond to a real top-level directory named "customers" in the Web server document root directory. The files that make up this application must all exist within the Web server document root directory. The global.asa file, if present, must be located in the top-level directory.

The ASP Server does not dynamically recognize ASP applications that are defined in the Sun Chili!Soft ASP configuration file, casp.cnfg, or that are defined by using an alias in the Web server configuration files. The ASP Server must be restarted to recognize them. ASP applications defined in the casp.cnfg file or by creating an alias in the Web server configuration files can include files outside of the Web server document root directory. The global.asa file, if present, must be located in the top-level directory referenced by the ASP application.

If there are naming conflicts between ASP applications that are defined in different directories, the ASP Server honors application definitions in the following order:

1. Web server aliases
2. casp.cnfg file entries
3. FrontPage services.cnf file entries
4. ASP User Configuration file entries

Note

Sun Chili!Soft ASP for UNIX- and Linux-based systems dynamically recognizes ASP applications created by FrontPage, but only if the application is not in a nested sub-Web. If the application (and its associated global.asa file) is located in a directory that is not a top-level directory of the Web server document root directory, you must define this application using either the `[applications]` section of Sun Chili!Soft ASP casp.cnfg file, or by adding an alias to your Web server configuration. For more information, see "Editing the Sun Chili!Soft Configuration File" in this chapter.

Defining an Application on iPlanet Web Server

For the purpose of defining Application and Session scope, the ASP Server considers all .asp files located in a virtual directory to be part of one application. You can use the **NameTrans** parameter in the obj.conf file to define an application. The following example defines an application called "/dosperros":

```
NameTrans fn="pfx2dir" from="/dosperros" dir="/opt/casp-
net30/caspsamp/dosperros"
```

If you are using the Web server's Administration tool, you can define an ASP application by adding an "additional document directory."

Defining an Application on Apache Web Server

For the purpose of defining Application and Session scope, the ASP Server considers all *.asp files located in a virtual directory to be part of one ASP application. You can use the **Alias** parameter in the srm.conf file (in httpd.conf for Apache 1.3.4, 1.3.6, or 1.3.9) to define an ASP application. The following example defines an application called "/caspsamp":

```
Alias /caspsamp "/[C_ASP_INSTALL_DIR]/samples"
```

where [C-ASP_INSTALL_DIR] is the directory in which Sun Chili!Soft ASP is installed.

If you have configured support for virtual hosts, you can define ASP applications on Apache Web Server as follows:

- By adding an entry to the `[applications]` section of casp.cnfg. This applies to the "real host" only.
- By adding an alias to the Web server configuration file (only if `use_aliases=yes` in the `[applications]` section of casp.cnfg.) If the alias appears outside a `<virtualhost> ... </virtualhost>` block, it applies to the "real host" only. If the alias appears inside a `<virtualhost> ... </virtualhost>` block, it applies to the virtual host.
- By adding an entry to the ASP User Configuration file. The name of this file is defined in the `[applications]` section of the casp.cnfg file. Sun Chili!Soft ASP looks for this file in the document root directory of each host, "real" or virtual. "Real host" entries apply to the "real host" only.
- By adding an entry to the services.cnf file generated by FrontPage. This file is located in the /_vti_pvt subdirectory of the root directory of each host, "real" or virtual. "Real host" entries apply to the "real host" only.

Relocating the System Files for a Shared Installation

For users installing Sun Chili!Soft ASP to shared file systems such as NFS or AFS (Andrew File System), use the following instructions to relocate the system files. If you are not installing to a shared file system, you should not alter the locations of the Sun Chili!Soft ASP system files.

Caution

> Take great care when making the changes described in this section. Changes you make could require a complete reinstall of Sun Chili!Soft ASP, and could void your eligibility for customer support. You should back up your data before making any changes.

Most of the configuration settings described in this section are easily accessed from the Sun Chili!Soft ASP Administration Console. It is strongly recommended that you use the Administration Console whenever possible, as described in "Using the Administration Console" in this chapter.

Relocating the Registry File

One of the key Sun Chili!Soft ASP system files, the registry file (registry.bin), must be located on a local file system that supports file locking to ensure proper operation. During Sun Chili!Soft ASP installation, the installer creates a script named chsetup.sh in the Sun Chili!Soft ASP installation directory (/opt/casp by default). Contained within this file is a line of code that resembles the following:

```
MWREGISTRY=[path name]/registry.bin
```

where `[path name]` is the current location of the registry file.

To relocate the registry file

1. Write down the current value of `[path name]`.
2. Create the new directory where you wish to relocate the registry file. This directory must reside on the local machine where the file system supports file locking. For the following steps, this new directory will be referred to as `<new path name>`.
3. Edit chsetup.sh with a text editor such as vi and change the following line:

   ```
   MWREGISTRY=[path name]/registry.bin
   ```

   to

   ```
   MWREGISTRY=[new path name]/registry.bin
   ```

4. Copy the registry file from its old location (`[path name]/registry.bin`) to its new location (`[new path name]/registry.bin`).

Relocating Sun Chili!Soft ASP PID Files

For users who want to install Sun Chili!Soft ASP to a shared file system, but move writeable Sun Chili!Soft ASP files to a local file system, Sun Chili!Soft ASP provides a mechanism to allow for this. For single machine Sun Chili!Soft ASP installations, this is not required, but has the added benefit that it may decrease network congestion.

There are three attributes of importance in the casp.cnfg file (which is contained under each `asp-<server>-<port>` directory). These are the **hashobj_pid** and **logfile** attributes (located in the `[machines]` section), and the **caspd_pid** attribute (located in the `[default machine]` section). These attributes allow you to specify the locations of the two process ID (PID) files. If you need to relocate these files, remember the following:

- If you have several Sun Chili!Soft ASP installations on a single physical server (with separate `asp-<server>-<port>` directories), then the casp.cnfg file in each directory may point to common directories for the PID and log files, but must point to different file names for the PID and log files.

For example, suppose the `[machines]` and `[default machine]` sections of your current casp.cnfg file resemble the following.

```
[machines]
count=1
machine1=127.0.0.1
portnumber=3000
logfile=/opt/casp/logs/server-3000
mtengine=0
disablerestart=0
hashobj_pid=/opt/casp/logs/asp-apache-3000/hashobj.pid

[default machine]
caspd_pid=/opt/casp/logs/asp-apache-3000/caspd.pid
maxprocesses=1
inherit_user=1
#user=nobody
#group=nobody
```

In this situation, to relocate all of your files off of the shared `/opt` directory to a `/usr/local/casp` directory, use the following procedure.

To relocate all files

1. Create the destination directory(ies), for example:

   ```
   mkdir -p /usr/local/casp/pids
   mkdir -p /usr/local/casp/logs
   ```

2. Edit the `casp.cnfg` file to resemble the following example:

   ```
   [machines]
   count=1
   machine1=127.0.0.1
   portnumber=3000
   logfile=/usr/local/casp/logs/server-3000
   mtengine=0
   disablerestart=0
   hashobj_pid=/usr/local/casp/pids/hashobj-3000.pid
   ```

```
[default machine]
caspd_pid=/usr/local/casp/pids/caspd-3000.pid
maxprocesses=1
inherit_user=1
#user=nobody
#group=nobody
```

3. Copy the server log file and the PID files to the directories you created.

Chapter 4: Building a Sun Chili!Soft ASP Application

Active Server Pages (ASP) enables the Web author and developer to easily create dynamic Web applications by using scripts that run on the Web server. An ASP page can contain a combination of HTML text, server-side scripts, and client-side scripts, creating an engaging experience for the Web user.

Sun Chili!Soft ASP enables the scripting logic to interface with five built-in ASP objects, which automatically handle many menial tasks, making application development easier. In addition to using these basic elements, you can extend ASP by using the Component Object Model (COM). This enables you to add sophisticated functionality by using components written in programming languages such as Java. You can incorporate this functionality into your Web applications by using scripts as the "glue" to link the COM objects. For example, Sun Chili!Soft ASP includes an ActiveX Data Object (ADO) component that provides a high-performance interface between Web pages and databases that adhere to the Open Database Connectivity (ODBC) standard. In addition, Chili!Beans support included with Sun Chili!Soft ASP enables you to use Java objects with your ASP applications.

The first section in this chapter discusses the basics of building a Sun Chili!Soft ASP application: creating an ASP page, adding server-side scripts and server-side includes, and defining the application. Next, the chapter discusses extending applications by using objects and components, connecting to databases, and developing applications to publish in locales other than the United States. Finally, it gives instructions for publishing a Sun Chili!Soft ASP application.

Who should read this chapter: Web authors and developers who are new to ASP and system administrators who want a basic introduction to Sun Chili!Soft ASP technology should read the first section. Web authors and developers who have mastered the basics of building ASP applications might be interested in reading the more advanced topics covered in subsequent sections. Everyone should read the last section about publishing a Sun Chili!Soft ASP application.

In this chapter:

Creating the Basic ASP Application

Sun Chili!Soft ASP combines ASP technology with server-side object-oriented components to provide an integrated Web development environment. A Sun Chili!Soft ASP application consists of the following elements:

- ASP files, or pages, which are plain text files having an .asp extension. ASP pages comprise a combination of standard HTML and scripting, written in either JScript or Visual Basic Scripting Edition (VBScript)
- Optional components written in any language
- An optional global.asa file that contains global application information and session information for individual users

Web servers normally send HTML files directly to the client's Web browser in response to HTTP requests. When a browser requests an ASP page, the Web server calls the Sun Chili!Soft ASP Server to read through the file. The ASP Server executes the server-side scripts and commands in the page, executes any components called by the scripts, and sends the resulting HTML page to the browser.

This section discusses the following steps that you can take to create a basic ASP application:

- Choosing an Authoring Tool
- Creating an ASP Page
- Adding Scripts
- Changing the Scripting Language
- Using Server-side Includes
- Defining the ASP Application
- Using the global.asa File

When you are ready to publish your application, be sure to read "Publishing a Sun Chili!Soft ASP Application" in this chapter.

Once you have mastered the basics in this section, you also might want to read the following advanced topics:

- Using Sun Chili!Soft ASP Built-in Objects
- Using Sun Chili!Soft ASP Installed Components
- Using Java Objects and Classes
- Connecting to a Database
- Developing International Applications

Choosing an Authoring Tool

ASP is one of the few technologies that can be used effectively for creating both sophisticated and entry-level Web applications. Because of the flexibility of ASP, there are many ASP development tools available for Web developers and authors with varying skill levels.

You can use any text editor to create *.asp files. As you progress, you may find it more productive to use an editor with enhanced support for ASP, such as Macromedia Dreamweaver UltraDev, Adobe GoLive, Microsoft FrontPage, or Microsoft Visual InterDev.

Sun Chili!Soft ASP, combined with one or more of these development tools, gives you a common Web application platform for:

- Applications that are large and small, simple or complex.
- A host of popular Web servers and operating systems.
- Developers and page creators with widely varying skill levels.

Note

Sun Chili!Soft ASP enables you to run ASP pages generated by a variety of development tools. Questions about the installation, configuration, and use of these tools should be directed to the tool's manufacturer.

Creating an ASP Page

The first step in building an ASP application is to create an ASP page. ASP pages are simply plain text files that have an .asp filename extension. An ASP page contains optional text (usually HTML and/or client-side scripts) interspersed with one or more ASP script blocks for interpretation by the server.

Any valid HTML page can be a valid ASP page, enabling the Web developer to easily transform a static Web site into a dynamic Web site by adding ASP scripts to existing HTML pages. With Sun Chili!Soft ASP, you can write scripts in VBScript or JScript. Saving the page with an *.asp filename extension tells the Web server how to process the script commands.

See also:

Creating the Basic ASP Application in this chapter

Adding Scripts in this chapter

Using Server-side Includes in this chapter

Using @ Directives in this chapter

Adding Scripts

Once you have created an ASP page, as described in "Creating an ASP Page" in this chapter, you can use a text editor or other authoring tool to insert script commands into the page. ASP pages can include both client-side scripts, which are processed by the browser, and server-side scripts (or ASP scripts), which are processed by the ASP Server.

ASP scripts enable you to dynamically create HTML responses based on the user's identity, parameters in the HTTP request, and interactions with other objects, such as ASP built-in objects, components, and databases. ASP enables you to assign values to variables, request information from the server, or combine any set of commands into procedures.

For example, a common use of Web applications is to process a form submitted by a browser. With ASP, you can embed scripts directly into an HTML file to process the form. The ASP Server processes the HTML and script commands and returns the results to the browser.

Within the ASP page, script blocks are set off from other text by using delimiters. You must use different script delimiters to distinguish between client-side and server-side scripts. You enclose client-side scripts between the `<script>` and `</script>` tags. You enclose server-side scripts between the delimiters `<%` and `/%>`.

You can write ASP scripts in either VBScript or JScript. The default scripting language for Sun Chili!Soft ASP is VBScript, but you can specify the scripting language for each ASP page. Your system administrator can also change the default scripting language for the ASP Server. For more information, see "Changing the Scripting Language" in this chapter.

As the ASP Server processes each ASP script block, it creates HTML text that is returned to the browser for rendering. Unlike client-side scripts, with server-side ASP scripts you do not need to be concerned about the capabilities of the browser; all processing is done at the server and only standard HTML is returned.

Users cannot copy server-side scripts because only the output is returned to the browser. Consequently, to view the results of a script you have added to an ASP page, you must first publish the page to an ASP Server and then request the page by using a Web browser.

The following example shows how you can combine standard HTML tags with a simple script that provides the current time of day:

```
<%@ LANGUAGE="VBSCRIPT" %>

<HTML>

<HEAD>

<META NAME="GENERATOR" Content="Microsoft Visual InterDev 1.0">

<META HTTP-EQUIV="Content-Type" content="text/html; charset=iso-8859-1">

<TITLE>An ASP Page</TITLE>

</HEAD>

<BODY>

<The time is now <%Response.Write Now%>

</BODY>

</HTML>
```

Both VBScript and JScript support the If-Then-Else construct, enabling you to embed some real logic into your HTML. The following example shows how you can set the greeting shown based upon the time of day:

```
<%If Time >= #12:00:00 AM# And Time < #12:00:00 PM# Then%>

Good Morning!

<%Else%>

Hello!

<%End If%>
```

See also:

Creating the Basic ASP Application in this chapter

Creating an ASP Page in this chapter

Using Server-side Includes in this chapter

Using @ Directives in this chapter

Changing the Scripting Language

Sun Chili!Soft ASP provides both VBScript and JScript script interpreters to process the commands in an ASP script. The default scripting language is VBScript, but you can change this to JScript for an ASP page by using the `<%@ LANGUAGE>` directive at the beginning of your ASP file, as described in "Using @ Directives" in this chapter.

You can change the scripting language for a single block of script by enclosing the block in `<SCRIPT> ... </SCRIPT>` tags. Normally, a block of code enclosed in `<SCRIPT>` tags runs on the client side, but you can force the block to run on the server by including the `runat=server` attribute, as shown in the following example:

```
<SCRIPT language=JScript runat=server>
```

Alternatively, your system administrator can change the default scripting language to either VBScript or JScript on the ASP Server. For Windows systems, see "Editing the Windows Registry" in "Chapter 3: Managing Sun Chili!Soft ASP." For UNIX and Linux systems, see "Editing the Sun Chili!Soft ASP Configuration File" in "Chapter 3: Managing Sun Chili!Soft ASP."

Using @ Directives

Sun Chili!Soft ASP provides `@` directives in addition to the available scripting commands. The following standard ASP `@` directives are available:

ENABLESESSIONSTATE. This directive turns session tracking on and off for an ASP page. If your page does not rely on session information, turning session tracking off can decrease the time it takes Sun Chili!Soft ASP to process the script. By default, sessions are enabled. For more information, see "Managing User Sessions" in this chapter. The syntax for this directive is as follows:

Syntax: ENABLESESSIONSTATE @ Directive

```
<%@ ENABLESESSIONSSTATE=True|False %>
```

LANGUAGE. By default, the primary scripting language for Sun Chili!Soft ASP is VBScript, but this can be changed to JScript for each ASP page by using the <%@ LANGUAGE> directive at the beginning of your ASP file. The syntax and parameters are as follows:

Syntax: LANGUAGE @ Directive

```
<%@ LANGUAGE=scriptengine %>
```

There must be a space between the @ and the keyword. More than one keyword can be specified in a directive; each keyword/value pair must be separated by a space. Do not put spaces around the equal sign (=).

Parameters: LANGUAGE @ Directive

scriptengine

The script engine that should process the script.

Note

The following standard ASP directives are not implemented by Sun Chili!Soft ASP: CODEPAGE, LCID, and TRANSACTION.

For more information about configuring Sun Chili!Soft ASP language support, see "Developing International Applications" in this chapter.

See also:

Creating the Basic ASP Application in this chapter

Creating an ASP Page in this chapter

Using Server-side Includes in this chapter

Using Server-side Includes

You use a server-side include directive to import a file into an ASP page during processing. Any text file can be imported, or "included." The contents of the included file are placed on the page at the location of the server-side include directive.

You can include files that themselves contain included files. In the event of a "loop," in which the first file contains an included file that in turn includes the first file, ASP reports an error. Included files can also be ASP files; the results of an included ASP file are placed at the position of the **#include** statement. You cannot build a server-side include statement programmatically because ASP processes **#include** directives before processing any script.

The syntax for a server-side include is as follows:

```
<!--#INCLUDE VIRTUAL|FILE="filename"-->
```

To specify the path, use the `virtual` keyword to indicate a path name beginning with a virtual directory. Use the `file` keyword to indicate a relative path name that begins with the directory containing the include file. For example, if a file is in the directory Dir1, and the file header1.inc is in Dir1/Headers, the following code would insert header1.inc in your file:

```
<!--#INCLUDE FILE="Headers/header1.inc"-->
```

If the **EnableParentPaths** configuration setting is set to **yes**, you can also use the **File** parameter with `../` syntax to include a file from a parent (higher-level) directory. By default, **EnableParentPaths** is set to **no**. In this case, the **CreateObject** ("**Scripting.FileSystemObject**") calls generated in the global.asa file by FrontPage do not work. Your system administrator must change **EnableParentPaths** to **yes**, or you must change the code generated by FrontPage in the global.asa file to **Server.CreateObject** ("**Scripting.FileSystemObject**"). For more information, see "Configuring File System Access" in "Chapter 3: Managing Sun Chili!Soft ASP."

There is no real performance penalty for using server-side includes. ASP saves files in memory in a compiled form after processing them. Processing only occurs the first time a file is accessed.

Often after you edit an "included" file, the change does not show up in your ASP page. The ASP Server only picks up changes in an included file if it recompiles the page that contains the **#include** directive. You can force a recompile of this page by "touching" the file or by making a trivial change that updates the timestamp on the file.

Within an included ASP file, script commands and procedures must be entirely contained within the script delimiters `<%` and `%>`, the HTML tags `<SCRIPT>` and `</SCRIPT>`, or the HTML tags `<OBJECT>` and `</OBJECT>`. That is, you cannot open a script delimiter in an included ASP file, and then close the delimiter in the referencing file. The script or script command must be a complete unit.

See also:

Creating the Basic ASP Application in this chapter

Creating an ASP Page in this chapter

Using @ Directives in this chapter

Defining the ASP Application

An ASP application is synonymous with a directory structure. It represents a collection of files and virtual directories that are intended to work together to create a Web-based application. An

application is defined by flagging a directory as the application start point. The application scope then includes all items within the directory and the sub-directories, except those that are included in another application. Applications can include a global.asa file that contains global application information and user session information.

For the Sun Chili!Soft ASP Server to recognize and process an ASP application, your administrator must first define the application on the server, as described in "Configuring ASP Applications" in "Chapter 3: Managing Sun Chili!Soft ASP."

See also:

Using the global.asa File in this chapter

Creating the Basic ASP Application in this chapter

Creating an ASP Page in this chapter

Using the global.asa File

Global.asa is an optional file that stores script procedures and objects used globally by an application. There can only be one global.asa file per ASP application. The file must be named global.asa and must be stored in the root directory of the application. The root directory of an application is the top-level directory containing all of the application files and sub-directories.

The only script procedures you can declare in a global.asa are the following:

- **Application_OnStart** event, as described in "Specifying Application Events" in this chapter.
- **Application_OnEnd** event, as described in "Specifying Application Events" in this chapter.
- **Session_OnStart** event, as described in "Managing User Sessions" in this chapter.
- **Session_OnEnd** event, as described in "Managing User Sessions" in this chapter.

Notes

Scripts that do not have session or application scope cause the server to return an error. Global.asa files that do not specify Application and Session events are ignored.

Script procedures declared in the global.asa file cannot be called from ASP pages in an application.

See also:

Saving Changes to the global.asa File in this chapter.

Specifying Application Events

An ASP application starts the first time the Web server receives a request for one of the ASP pages contained in the application directory. The application ends when the Web server is shut down. You can create global data for an application using the built-in ASP **Application** object.

You can assign variables and object instances to application variables so that they are available to all pages of an application.

When an application starts, the **Application_OnStart** event occurs. The **Application_OnEnd** event occurs when the application shuts down. When the application starts or stops, the server looks in the global.asa file to find the event scripts.

ASP Application_OnStart Event

The **Application_OnStart** event occurs before the first session is created, before the **Session_OnStart** event occurs. Only the **Server** and **Application** built-in objects are available. Referencing the **Session, Request**, or **Response** object in the **Application_OnStart** event generates an error.

Syntax: ASP Application_OnStart Event

```
<SCRIPT LANGUAGE=ScriptLanguage RUNAT=Server>

Sub Application_OnStart

. . .

End Sub

</SCRIPT>
```

Parameters: ASP Application_OnStart Event

ScriptLanguage

Specifies the scripting language used to write the event script, either VBScript or JScript. If more than one event uses the same scripting language, the events can be enclosed within a single set of <SCRIPT> tags.

Runat

Must be "Server."

ASP Application_OnEnd Event

The **Application_OnEnd** event occurs when the application ends, after the **Session_OnEnd** event (see below). Only the **Server** and **Application** objects are available.

Syntax: ASP Application_OnEnd Event

```
<SCRIPT LANGUAGE=ScriptLanguage RUNAT=Server>

Sub Application_OnEnd

. . .

End Sub

</SCRIPT>
```

Parameters: ASP Application_OnEnd Event

ScriptLanguage

Specifies the scripting language used to write the event script, either VBScript or JScript. If more than one event uses the same scripting language, the events can be enclosed within a single set of `<SCRIPT>` tags.

Runat

Must be "Server."

See also:

Managing User Sessions in this chapter

Using the global.asa File in this chapter

Saving Changes to the global.asa File in this chapter

Managing User Sessions

Communication between a browser and a Web server uses the HTTP protocol. This protocol is stateless, meaning that there is no information retained when a user goes from one page to another. In a real-world application, it is usually necessary to retain information between pages visited by the same user. The use of an application by a single user is called a session.

Sun Chili!Soft ASP includes the built-in **Session** object for retaining session information. When a new session starts, an instance of the **Session** object is created automatically. You can assign variables and object instances to session variables so that they are available to all pages of the application visited by the same user.

The global.asa file can contain the following handlers for two session-level events: **Session_OnStart** and **Session_OnEnd**, which are described later in this topic. These subroutines are automatically executed the first time a user accesses an ASP page within the application. Sessions exist until one of the following occurs:

- The user closes the browser
- The session times out (configurable by the ASP developer)
- The session is explicitly abandoned (**Session.Abandon**)

Turning Session Tracking On and Off

The ENABLESESSIONSTATE @ directive turns off session tracking for a page. If your page does not rely on session information, turning off session tracking can decrease the time it takes Sun Chili!Soft ASP to process the script. By default, sessions are enabled.

Syntax

```
<%@ ENABLESESSIONSSTATE=True|False %>
```

ASP Session_OnStart Event

The **Session_OnStart** event occurs when the server creates a new session. The server processes this script prior to executing the requested page. The **Session_OnStart** event is where, when you set any session-wide variables, those variables will be set before any pages are accessed. All of the built-in objects are available and can be referenced in the **Session_OnStart** event script.

Syntax: Session_OnStart Event

```
<SCRIPT LANGUAGE=ScriptLanguage RUNAT=Server>

Sub Session_OnStart

. . .

End Sub

</SCRIPT>
```

Parameters: Session_OnStart Event

ScriptLanguage

Specifies the scripting language used to write the event script, either VBScript or JScript. If more than one event uses the same scripting language, the events can be enclosed within a single set of <SCRIPT> tags.

Runat

Must be "Server."

ASP Session_OnEnd Event

The **Session_OnEnd** event occurs when a session is abandoned or times out. Only the **Application**, **Server** and **Session** built-in objects are available.

Syntax: ASP Session_OnEnd Event

```
<SCRIPT LANGUAGE=ScriptLanguage RUNAT=Server>

Sub Session_OnEnd

. . .

End Sub

</SCRIPT>
```

Parameters: ASP Session_OnEnd Event

ScriptLanguage

Specifies the scripting language used to write the event script, either VBScript or JScript. If more than one event uses the same scripting language, the events can be enclosed within a single set of <SCRIPT> tags.

Runat

Must be "Server."

See also:

Specifying Application Events in this chapter

Using the global.asa File in this chapter

Saving Changes to the global.asa File in this chapter

Saving Changes to the global.asa File

When you make changes to the global.asa file and then save them, the Sun Chili!Soft ASP Server reloads and compiles the file. Saving changes to a file "included" by the global.asa file does not force the server to reload it, so you must force a reload by resaving the global.asa file.

Note that the ASP Server processes all current application requests before it recompiles. During that time, additional requests are refused, returning the message, "The request cannot be processed while the application is being restarted."

See also:

Using the global.asa File in this chapter

Specifying Application Events in this chapter

Managing User Sessions in this chapter

Using Sun Chili!Soft ASP Built-in Objects

Sun Chili!Soft ASP includes five built-in objects--**Application**, **Request**, **Response**, **Server**, and **Session**--that handle many common programming tasks. These objects enable you to avoid much of the overhead associated with complex Web programming, so you can focus on creating interesting, interactive Web content rather than on doing low-level programming.

Sun Chili!Soft ASP objects use "methods" to perform some type of procedure and "properties" to store object attributes (such as color, font, or size). The **Request** and **Response** objects also contain "collections" (bits of information that are accessed in the same way).

For Web applications requiring more powerful programming, you can use Component Object Model (COM) objects to process data and deliver output, with scripts acting as the "glue" to link COM objects. Sun Chili!Soft ASP also supports JavaBeans, Enterprise JavaBeans (EJB), and Common Object Request Broker Architecture (CORBA) objects. For more information, see "Component Programmer's Reference" in "Chapter 5: Developer's Reference."

This section provides a basic overview of the following built-in ASP objects included with Sun Chili!Soft ASP. For more information, see "ASP Built-in Objects Reference" in "Chapter 5: Developer's Reference."

In this section:

- ASP Application Object Overview: shares application-level information and control settings for the lifetime of the application
- ASP Request Object Overview: gets information from the user
- ASP Response Object Overview: sends information to the user
- ASP Server Object Overview: controls the Web server
- ASP Session Object Overview: stores information about and changes settings for the user's current session

ASP Request Object Overview

The **Request** object is one of the five Sun Chili!Soft ASP built-in objects. The **Request** object is used to get information from the user that is passed along in an HTTP request. Both the **Request** and **Response** objects support the following collections:

- **QueryString**: gets text
- **Form**: gets data from an HTML form
- **Cookies**: gets the value of application-defined cookie
- **ServerVariables**: gets HTTP information, such as the server name

See also:

Using Sun Chili!Soft ASP Built-in Objects in this chapter

ASP Request Object in "Chapter 5: ASP Built-in Objects Reference"

ASP Response Object Overview

The **Response** object is one of the five Sun Chili!Soft ASP built-in objects. The **Response** object is used to send information to the user. The **Response** object supports only **Cookies** as a collection (to set cookie values). Both the **Request** and **Response** objects support the following collections:

- **QueryString**: gets text, such as a name
- **Form**: gets data from an HTML form
- **Cookies**: gets the value of application-defined cookie
- **ServerVariables**: gets HTTP information, such as the server name

The **Response** object also supports a number of properties and methods. Supported properties are:

- **Buffer**: buffers page output at the server. When this is set to TRUE, the server will not send a response until all of the server scripts on the current page have been processed, or until the **Flush** or **End** method has been called.
- **ContentType**: sets the type of content (such as text/HTML, Excel, and so forth).

- **Expires**: sets the expiration based on minutes (when the data in the user's cache for this Web page is considered invalid; for example, expires in 10 minutes).
- **ExpiresAbsolute**: enables you to set the expiration date to an absolute date and time.
- **Status**: returns the status line (defined in the HTTP specification for the server).

Supported methods are:

- **AddHeader**: adds an HTML header with a specified value
- **AppendToLog**: appends a string to the end of the Web server log file
- **BinaryWrite**: writes binary data (for example, Excel spreadsheet data)
- **Clear**: clears any buffered HTML output
- **End**: stops processing of the script
- **Flush**: sends all of the information in the buffer
- **Redirect**: redirects the user to a different URL
- **Write**: writes into the HTML stream. You can do this by using the construct **Response.Write**("hello") or the shortcut command `<%="hello"%>`.

For information about using the ASP built-in objects, see "ASP Built-in Objects Reference" in "Chapter 5: Developer's Reference."

See also:

Using Sun Chili!Soft ASP Built-in Objects in this chapter

ASP Response Object in "Chapter 5: ASP Built-in Objects Reference"

ASP Server Object Overview

The **Server** object is one of the five Sun Chili!Soft ASP built-in objects. The **Server** object provides high-level access to the ASP Server. Along with the **Application** object, the **Server** object provides ASP applications with global data: information that applies to all users of the application. The **Server** object gives you programmatic control of the Web server, providing access to HTTP services that you would otherwise need to code for each application. By using **Server** object properties and methods, you can create objects, execute scripts on other ASP pages, translate virtual path names to physical path names, and perform server-side redirects.

The **Server** object supports one property, **ScriptTimeout**, which allows you to set the value for when the script processing will time out, and the following methods:

- **CreateObject**: creates an instance of a server component. This component can be any component that you have installed on your server (such as an ActiveX).
- **HTMLEncode**: encodes the specified string in HTML.
- **MapPath**: maps the current virtual path to a physical directory structure. You can then pass that path to a component that creates the specified directory or file on the server.

- **URLEncode**: applies URL encoding to a specified string.

For information about using the ASP built-in objects, see "ASP Built-in Objects Reference" in "Chapter 5: Developer's Reference."

See also:

Using Sun Chili!Soft ASP Built-in Objects in this chapter

ASP Server Object in "Chapter 5: ASP Built-in Objects Reference"

ASP Session Object Overview

The **Session** object is one of the five Sun Chili!Soft ASP built-in objects. The **Session** object is used to store information about the current user's session. Variables stored with this object exist as long as the user's session is active, even if more than one application is used.

This object supports one method, **Abandon**, which abandons the current Web server session, destroying any objects. It supports two properties, **SessionID**, containing the identifier for the current session, and **Timeout**, specifying a time-out value for the session. Keep in mind that the session identifier persists as long as the current Web server session is running. If you shut down the Web server service, the identifiers restart. Therefore, it is not a good idea to use the session identifier to create logon IDs, because you could end up with duplicates.

For information about using the ASP built-in objects, see "ASP Built-in Objects Reference" in "Chapter 5: Developer's Reference."

See also:

Using Sun Chili!Soft ASP Built-in Objects in this chapter

ASP Session Object in "Chapter 5: ASP Built-in Objects Reference"

ASP Application Object Overview

The **Application** object is one of the five Sun Chili!Soft ASP built-in objects. The **Application** object stores information that persists for the entire lifetime of an ASP application, which is generally the entire time that the Web server is running.

The **Application** object is a good place to store information that must exist for more than one user (such as a page counter). However, because a new instance of this object is not created for each user, errors that might not show up when the code is called once might show up when it is called many times in a row. In addition, because all users share the **Application** object, it can be difficult to implement threading.

For more information about using the ASP built-in objects, see "ASP Built-in Objects Reference" in "Chapter 5: Developer's Reference."

See also:

Using Sun Chili!Soft ASP Built-in Objects in this chapter

ASP Application Object in "Chapter 5: ASP Built-in Objects Reference"

Using Sun Chili!Soft ASP Installed Components

In addition to the five built-in objects described in "Using Sun Chili!Soft ASP Built-in Objects," Sun Chili!Soft ASP automatically installs a number of components that you can use to build dynamic Web pages. The following table lists these installed components. For more information about using them, see "ASP Component Reference" in "Chapter 5: Developer's Reference."

Component	Description
Ad Rotator	Creates an **AdRotator** object that automates the rotation of advertisement images on a Web page.
Browser Capabilities	Creates a **BrowserType** object that determines the type, version, and capabilities of every browser that visits your site.
Content Linking	Creates a **NextLink** object that manages a list of URLs so that you can treat the pages in your Web site like the pages in a book.
Content Rotator	Creates a **ContentRotator** object that automatically rotates HTML content strings on a Web page.
Counters	Creates a **Counters** object that can create, store, increment, and retrieve any number of individual counters.
Database Access	Provides access to databases using ActiveX Data Objects (ADO).
FileSystemObject Object (VBScript)	Provides access to file input and output.
FileSystemObject Object (JScript)	
MyInfo	Creates a **MyInfo** object that keeps track of personal information, such as the site administrator's name, address, and display choices.
Tools	Creates a **Tools** object that provides utilities that enable you to easily add sophisticated functionality to your Web pages.

Using Java Objects and Classes

Through Chili!Beans, Sun Chili!Soft ASP provides support for Java objects and classes. Chili!Beans is an ActiveX control that acts as a wrapper to enable Java objects to be used by

Component Object Model (COM) controllers such as ActiveX scripting engines or VBScript. Chili!Beans is designed to work with Java Virtual Machine (JVM) versions 1.2 or greater.

For more information, see "Chili!Beans Component Reference" in "Chapter 5: Developer's Reference."

See also:

Using Sun Chili!Soft ASP Installed Components in this chapter

ASP Component Reference in "Chapter 5: Developer's Reference"

Connecting to a Database

The topics in this section take you through the two basic steps required for connecting to a database from an ASP page: creating a connection string, and opening a database connection. This section also explains how to use FrontPage database features with Sun Chili!Soft ASP. It also explains how to migrate a Microsoft Access database running on a Windows-based computer to a dBASE database running on a UNIX or Linux system.

A connection string provides information required by the Sun Chili!Soft ASP Server to establish the connection. Within a connection string, you can use one of three ways to specify information about a database, as described later in this section: a system DSN, a DSN-less connection string, and a file DSN.

You open a database connection by using the ADO **Connection** object included with Sun Chili!Soft ASP. You can then use other ADO objects to display and manipulate data on the ASP page. For more information about using ADO objects, see "ADO Component Reference" in "Chapter 5: Developer's Reference."

If you are going to pass data exceeding 64,000 bytes to a database, your system administrator should increase the **maxLongFieldLength** parameter for ADO, as described in "Editing the Chili!Soft Configuration File" in "Chapter 3: Managing Sun Chili!Soft ASP."

In this section:

- Creating Connection Strings
- Opening the Database Connection
- Using FrontPage Database Features
- Migrating a Microsoft Access Database to dBASE

Creating Connection Strings

When you want to connect to a database from an ASP page, your first step is to create the connection string. This provides information (in the form of parameters and their values) that is required for the server to establish the connection.

Each type of database has a specific set of parameters for which you must specify values; these are the required parameters. Some databases also provide optional parameters that you can specify to implement special features.

Exactly what you must include in a connection string depends on the type of database and the approach you use to specify its parameters. Sun Chili!Soft ASP supports the following three approaches to specifying parameters in a connection string:

- System DSNs. With a system DSN, all you need to provide in the connection string is the name of the DSN that your system administrator has configured for the database on the ASP Server. For more information, see "Using System DSNs" in this section.
- File DSNs. A file DSN is similar to a system DSN, except the database information is contained in a file (*.dsn) that can be stored in the root directory for a virtual host, rather than being stored centrally by the ASP Server. File DSNs are useful in shared Web hosting environments because a system administrator does not need to configure each file DSN; users can configure their own. For more information, see "Using File DSNs" in this section.
- DSN-less connection strings. With a DSN-less connection string, you specify all of the required database information in the connection string. For more information, see "Using DSN-less Connection Strings" in this section.

Note about using Windows connection strings with Sun Chili!Soft ASP for UNIX or Linux

Connection strings must be constructed according to the requirements of the ODBC driver being used. Sun Chili!Soft ASP for Windows uses standard Windows ODBC drivers, so connection strings you developed for Windows will work. However, the ODBC drivers for UNIX and Linux platforms are different than for Windows, so before you can use Windows connection strings with Sun Chili!Soft ASP for UNIX or Linux, you must edit them to use the syntax described in this section.

When creating file system references in ASP applications, keep in mind that UNIX and Linux are case-sensitive operating systems. Be sure to use the correct capitalization in all references to files and directories.

Ask your server administrator which approach you should use in your specific Web server environment.

Once you have created the connection string in your ASP page, you can add the code needed to open a database connection, as described in "Opening the Database Connection" in this chapter.

Note

On UNIX and Linux systems, Sun Chili!Soft ASP installs the ODBC drivers to support a number of databases. However, it does not support all databases on all platforms. To see the list of installed drivers for your platform, go to the platform-specific installation requirements section in "Installing and Uninstalling Sun Chili!Soft ASP" in "Chapter 2: Installing and Configuring Sun Chili!Soft ASP."

Using System DSNs

As discussed in "Creating Connection Strings" in this chapter, using a system DSN is one way to specify database information in a connection string.

Before you can use a system DSN in a connection string, your administrator must first add it to the ASP Server, as described in "Adding a DSN" in "Chapter 3: Managing Sun Chili!Soft ASP." This saves information on the ASP Server about all parameters required for connecting to the database.

Note

On UNIX and Linux systems, Sun Chili!Soft ASP installs the ODBC drivers to support a number of databases. However, it does not support all databases on all platforms. To see the list of installed drivers for your platform, go to the platform-specific installation requirements section in "Installing and Uninstalling Sun Chili!Soft ASP" in "Chapter 2: Installing and Configuring Sun Chili!Soft ASP."

Once a system DSN is configured, rather than needing to specify all of the database information in the connection string as you do with DSN-less connection strings, you can simply reference the system DSN name. When you do this, the ASP Server uses the information stored in the system DSN to establish the connection.

Often, all you need to provide in the connection string is the name of the DSN that your system administrator has configured for the database. In this case, use the following syntax:

```
connect_string = "dsn=[dsn_name]"
```

where `[dsn_name]` is the name your system administrator defined for the DSN.

However, if the username and password required for connecting to the database are not specified in the system DSN, you must include them in the connection string. Ask your database administrator for this information. Be sure to use the correct syntax for your type of database, as follows:

```
connect_string = "dsn=[dsn_name]; UID=[username]; PWD=[password]"
```

Note

dBASE does not require a username and password.

If your system administrator asks you to use file DSNs or DSN-less connection strings rather than system DSNs, see "Using File DSNs" and "Using DSN-less Connection Strings" in this chapter. However, you must use system DSNs for connecting to Microsoft Access and Microsoft SQL Server 6.5 databases. You cannot use DSN-less connection strings or file DSNs for connecting to these databases.

See also:

Connecting to a Database in this chapter

Using FrontPage Database Features in this chapter

Using DSN-less Connection Strings

As discussed in "Creating Connection Strings" in this chapter, using a DSN-less connection string is one way to specify the information (in the form of parameters and their values) that is needed for establishing a database connection. Unlike system DSNs and file DSNs, which incorporate

this information by reference, DSN-less connection strings include all required database parameters.

You use the following syntax for a connection string:

```
connect_string = "[parameter_1=value_1; parameter_2=value_2; parameter_3=value_3]"
```

where `[parameter_1=value1; parameter_2=value_2; parameter_3=value_3]` specifies the required parameters for the given database.

The following example shows a DSN-less connection string for a MySQL database:

```
connect_string = "Driver={Mysql}; Server=[server_name]; Database=[database_name];
UID=[username]; PWD=[password]"
```

where `[server_name]` is the name of the database server, `[database_name]` is the name of the database, and `[username]` and `[password]` are the username and password required for accessing the database.

Different types of databases can require that you specify different parameters. The parameters to configure for each database in a DSN-less connection string are provided in "Syntax for DSN-less Connection Strings" in this section.

Note about using Windows connection strings with Sun Chili!Soft ASP for UNIX or Linux

Connection strings must be constructed according to the requirements of the ODBC driver being used. Sun Chili!Soft ASP for Windows uses standard Windows ODBC drivers, so connection strings you developed for Windows will work. However, the ODBC drivers for UNIX and Linux platforms are different than for Windows, so before you can use Windows connection strings with Sun Chili!Soft ASP for UNIX or Linux, you must edit them to use the syntax described in this section.

Note about supported databases

With Sun Chili!Soft ASP for UNIX or Linux you cannot use DSN-less connection strings or file DSNs for connecting to Microsoft Access or Microsoft SQL Server 6.5 databases; you must use system DSNs for connecting to these databases.

On UNIX and Linux systems, Sun Chili!Soft ASP installs the ODBC drivers to support a number of databases. However, it does not support all databases on all platforms. To see the list of installed drivers for your platform, go to the platform-specific installation requirements section in "Installing and Uninstalling Sun Chili!Soft ASP" in "Chapter 2: Installing and Configuring Sun Chili!Soft ASP."

See also:

Connecting to a Database in this chapter

Creating Connection Strings in this chapter

Syntax for DSN-less Connection Strings

The following table lists the parameters to define and the syntax to use for each type of database in DSN-less connection strings.

Database Type	Syntax for DSN-less Connection Strings
DB2	`connect_string = "Driver={DB2}; IP=[ip_address]; Port=[port_number]; Database=[database_name]; UID=[username]; PWD=[password]"` where `[ip_address]` is the IP address of the database server, `[port_number]` is the port for the database server, `[database_name]` is the name of the database, and `[username]` and `[password]` are the username and password required for accessing the database.
dBASE 5	`connect_string = "Driver={Dbase}; DBQ=[pathname]; defaultDir=[default_directory]"` where `[pathname]` is the absolute path name of the directory containing the database file and `[default_directory]` is the default directory for the database.
Informix 7, 9	`connect_string = "Driver={Informix}; ServerName=[server_name]; Database=[database_name]; UID=[username]; PWD=[password]"` where `[server_name]` is the name of the database server, `[database_name]` is the name of the database, and `[username]` and `[password]` are the username and password required for accessing the database.
Informix 2000	`connect_string = "Driver={Informix}; HostName=[host_name]; ServerName=[server_name]; Port=[port_number];Database=[database_name]; UID=[username]; PWD=[password]"` where `[host_name]` is the name of the computer on which the database server resides, `[server_name]` is the name of the database server as it appears in the sqlhosts file, `[port_number]` is the port on which the database server is configured to listen, `[database_name]` is the name of the database, and `[username]` and `[password]` are the username and password required for accessing the database.
Microsoft Access	DSN-less connection strings and file DSNs are not supported for Microsoft Access databases. You must use system DSNs.
Microsoft SQL Server 6.5	DSN-less connection strings and file DSNs are not supported for Microsoft SQL Server 6.5 databases. You must use system DSNs.
Microsoft SQL Server 7.0 and 2000	`connect_string = "Driver={SQL Server}; Database=[database_name]; Address=[ip_address],[port_number]; UID=[username]; PWD=[password]"` where `[database_name]` is the name of the database,

`[ip_address],[port_number]` is the IP address of the database server and the port on which the database server is configured to listen, and `[username]` and `[password]` are the username and password required for accessing the database.

MySQL

```
connect_string = "Driver={Mysql};
Server=[server_name]; Database=[database_name];
UID=[username]; PWD=[password]"
```

where `[server_name]` is the name of the database server, `[database_name]` is the name of the database, and `[username]` and `[password]` are the username and password required for accessing the database.

Oracle 7, 8

```
connect_string = " Driver={Oracle};
Server=[TNS_name]; UID=[username];
PWD=[password]"
```

where `[TNS_name]` is the TNS name as defined in the tnsnames.ora file, and `[username]` and `[password]` are the username and password required for accessing the database.

Oracle 8i, 9i

```
connect_string = " Driver={Oracle};
Host=[host_name];Port=[port_number];
SID=[oracle_SID]; UID=[username]; PWD=[password]"
```

where `[host_name]` is the computer on which the database server resides, `[port_number]` is the port on which the database server is configured to listen, `[oracle_SID]` is the Oracle System Identifier that refers to the instance of Oracle running on the server, and `[username]` and `[password]` are the username and password required for accessing the database.

PostgreSQL

```
connect_string = " Driver={Postgres};
Server=[server_name]; Port=[port_number];
Database=[database_name]; UID=[username];
PWD=[password]"
```

where `[server_name]` is the name of the database server, [`port_number`] is the port on which the database server is configured to listen, `[database_name]` is the name of the database, and `[username]` and `[password]` are the username and password required for accessing the database.

Sybase

```
connect_string = " Driver={Sybase};
NetworkAddress=[host_name],[port_number];
Database=[database_name]; UID=[username];
PWD=[password]"
```

where `[host_name],[port_number]` is the IP address of the database server and the port on which the database server is configured to listen, `[database_name]` is the name of the database,; and

	`[username]` and `[password]` are the username and password required for accessing the database.
Text	`connect_string = " Driver={Text}; Database=[database_location]"` where `[database_location]` is the directory in which the text files are stored.

See also:

Using DSN-less Connection Strings in this chapter

Using File DSNs

As discussed in "Creating Connection Strings" in this chapter, using file DSNs is one way to specify the information needed for establishing a connection to a database from an ASP application. This topic explains how to create a file DSN and reference it from within a connection string.

When you have a number of connection strings referencing the same database, file DSNs can be quicker to implement than DSN-less connection strings. File DSNs can also make ASP applications easier to port from the development environment to the production server because you can edit the database information in a single file, rather than editing multiple connection strings.

To use file DSNs, the first step is to create a file containing the required parameters and values for the database with which you want to connect. Then you simply reference the file from within the connection string, rather than duplicating the database information each time.

To create a file DSN, open a plain text file and specify the parameters for the database to which you want to connect by using the following general syntax:

```
[ODBC]

a=b

c=d

e=f
```

where `a=b, c=d`, and `e=f` are the key-value pairs that define the database parameters and their values. One of the key-value pairs must specify the name of the ODBC driver for the database. The parameters you must configure for each database are provided in "Parameters for File DSNs" in this section.

Note about using Windows file DSNs with Sun Chili!Soft ASP for UNIX or Linux

File DSNs and connection strings must be constructed according to the requirements of the ODBC driver being used. On Windows, Sun Chili!Soft ASP uses the same ODBC drivers as Microsoft ASP, so you do not need to change any file DSNs or connection strings to use them. However, the ODBC drivers available for UNIX and Linux platforms are different than for Windows. To connect to a database from an ASP application that you developed for Windows on Sun Chili!Soft ASP for UNIX or Linux, you must edit your file DSNs and connection strings to use the syntax described in this topic.

Also, when porting file DSNs to UNIX or Linux systems, be sure to remove the "control-M" characters that Windows inserts at the end of each line.

The following example shows a file DSN for a Sybase 11 database:

```
[ODBC]

Driver={Sybase}

Server=Sun

DB=MyDatabase

UID=John

PWD=Some.Password
```

When finished defining parameters, give the file a DSN filename extension (*.dsn) and save it in the document root of your Web server or virtual host.

Once you have created the file DSN, you can refer to it from within a connection string. The syntax to use is as follows:

```
connect_string = "FileDSN=[MyFileDSN.dsn]"
```

- or -

```
connect_string = "File_Name=[MyFileDSN.dsn]"
```

where `[MyFileDSN.dsn]` is the absolute path name of the file DSN (*.dsn) containing the database parameters and values.

In a shared Web hosting environment, such as with an Internet Service Provider, you might not know the directory structure above the document root for your virtual host. In this situation, you cannot specify the absolute path name of the file DSN, so you must use the **Server.mapPath** directive instead. The following example uses a file DSN that is stored in the document root of the virtual host:

```
dim myConnFile,connection_string

myConnFile = Server.mapPath("/") & "/" & "MyFileDSN.dsn"

connect_string = "FileDSN=" & myConnFile
```

Note about supported databases

On UNIX and Linux systems, Sun Chili!Soft ASP installs the ODBC drivers to support a number of databases. However, it does not support all databases on all platforms. To see the list of installed drivers for your platform, go to the platform-specific installation requirements section in "Installing and Uninstalling Sun Chili!Soft ASP" in "Chapter 2: Installing and Configuring Sun Chili!Soft ASP."

Note about Microsoft Access and Microsoft SQL Server databases

You cannot use DSN-less connection strings or file DSNs for connecting to Microsoft Access or Microsoft SQL Server 6.5 databases from Sun Chili!Soft ASP for UNIX or Linux; you must use system DSNs.

See also:

Connecting to a Database in this chapter

Creating Connection Strings in this chapter

Using System DSNs in this chapter

Using FrontPage Database Features in this chapter

Parameters for File DSNs

The following table lists the parameters you must define in a file DSN for each type of database. In each case, use the driver name for your database that is provided in the table.

Database Type	Parameters

DB2	`Driver={DB2}`
	`IP=[ip_address]`
	`Port=[port_number]`
	`Database=[database_name]`
	`UID=[username]`
	`PWD=[password]`
	where `[ip_address]` is the IP address of the database server, `[port_number]` is the port for the database server, `[database_name]` is the name of the database, and `[username]` and `[password]` are the username and password required for accessing the database.

dBASE 5	`Driver={Dbase}`
	`DBQ=[pathname]`
	`defaultDir=[default_directory]`
	where `[pathname]` is the absolute path name of the directory containing the database file and `[default_directory]` is the default directory for the database.

Informix 7, 9	`Driver={Informix}` `Server=[server_name]`

```
Database=[database_name]
UID=[username]
PWD=[password]
```

where `[server_name]` is the name of the database server, `[database_name]` is the name of the database, and `[username]` and `[password]` are the username and password required for accessing the database.

Informix 2000

```
Driver={Informix}
HostName=[host_name]
Server=[server_name]
Port=[port_number]
Database=[database_name]
UID=[username]
PWD=[password]
```

where `[host_name]` is the name of the computer on which the database server resides, `[server_name]` is the name of the database server as it appears in the sqlhosts file, `[port_number]` is the port on which the database server is configured to listen, `[database_name]` is the name of the database, and `[username]` and `[password]` are the username and password required for accessing the database.

Microsoft Access

DSN-less connection strings and file DSNs are not supported for Microsoft Access databases. You must use system DSNs.

Microsoft SQL Server 6.5

DSN-less connection strings and file DSNs are not supported for Microsoft SQL Server 6.5 databases. You must use system DSNs.

Microsoft SQL Server 7.0 and 2000

```
Driver={SQL Server}
Address=[ip_address],[port_number]
Database=[database_name]
UID=[username]
PWD=[password]
```

where `[database_name]` is the name of the database; `[ip_address],[port_number]` is the IP address of the database server and the port on which the database server is configured to listen, and `[username]` and `[password]` are the username and password required for accessing the database.

MySQL

```
Driver={Mysql}
Server=[server_name]
Database=[database_name]
UID=[username]
```

```
PWD=[password]
```

where [server_name] is the name of the database server, [database_name] is the name of the database, and [username] and [password] are the username and password required for accessing the database.

Oracle 7, 8

```
Driver={Oracle}
Server=[TNS_name]
UID=[username]
PWD=[password]
```

where [TNS_name] is the TNS name as defined in the tnsnames.ora file, and [username] and [password] are the username and password required for accessing the database.

Oracle 8i, 9i

```
Driver={Oracle}
Host=[host_name]
Port=[port_number]
SID=[oracle_SID]
Server=[TNS_name]
UID=[username]
PWD=[password]
```

where [host_name] is the computer on which the database server resides, [port_number] is the port on which the database server is configured to listen, [oracle_SID] is the Oracle System Identifier that refers to the instance of Oracle running on the server, and [username] and [password] are the username and password required for accessing the database.

PostgreSQL

```
Driver={Postgres}
Server=[server_name]
Port=[port_number]
Database=[database_name]
UID=[username]
PWD=[password]
```

where [server_name] is the name of the database server, [port_number] is the port on which the database server is configured to listen, [database_name] is the name of the database, and [username] and [password] are the username and password required for accessing the database.

Sybase

```
Driver={Sybase}
NetworkAddress=[host_name],[port_number]
```

```
Database=[database_name]

UID=[username]

PWD=[password]
```

where `[host_name],[port_number]` is the IP address of the database server and the port on which the database server is configured to listen, `[database_name]` is the name of the database,; and `[username]` and `[password]` are the username and password required for accessing the database.

Text

```
Driver={Text}

Database=[database_name]
```

where `[database_location]` is the directory in which the text files are stored.

See also:

Using File DSNs in this chapter

Opening the Database Connection

Sun Chili!Soft ASP includes an ActiveX Data Object (ADO) control that developers can use for initializing connections to databases and for retrieving and manipulating data on a Web page. The ADO **Connection** object opens and closes database connections by using ODBC drivers. Other ADO objects act as containers for storing information that is passed to and from the database. The most common container is a **Recordset** object, which stores the results of a **SELECT** SQL query.

The topic, "Creating Connection Strings," explains the first step to take to connect an ASP page to a database. After creating the connection string, your next step is to use the ADO control included with Sun Chili!Soft ASP to open a database connection, as described in this topic.

Note

On UNIX and Linux systems, Sun Chili!Soft ASP installs the ODBC drivers to support a number of databases. However, it does not support all databases on all platforms. To see the list of installed drivers for your platform, go to the platform-specific installation requirements section in "Installing and Uninstalling Sun Chili!Soft ASP" in "Chapter 2: Installing and Configuring Sun Chili!Soft ASP."

To open a database connection, you first add code for creating an instance of the ADO **Connection** object, as shown in the following example:

```
set dbConn = server.createObject("ADODB.connection")
```

Next, you add code to call the **Connection** object **Open** method, which takes the **connect_string** parameter, as shown in the next example:

```
dbConn.open connect_string
```

This sends a request to the ODBC Manager to create an instance of the ODBC driver specified by the connection string that you previously created. ADO then passes the remainder of the connection string to the ODBC driver, which uses this information for connecting to the database.

Once you have established the connection, you can use the other ADO objects to retrieve, display, and manipulate data on your Web page, as described in "ADO Component Reference" in "Chapter 5: Developer's Reference."

If desired, you can also use FrontPage for creating connection strings and displaying data on a Web page, as described in "Using FrontPage Database Features" in this chapter.

Note

Before you create a database connection, it is recommended that you ask your system administrator to verify that the appropriate ODBC driver for your database is configured and functioning properly. It is a good idea to test the driver on a nonproduction server because a malfunctioning ODBC driver can bring down your ASP Server.

See also:

Connecting to a Database in this chapter.

Using FrontPage Database Features

This section describes Sun Chili!Soft ASP support for the database connectivity features of FrontPage.

In this section:

- Using FrontPage Database Connections
- Displaying Data on a Web Page with FrontPage

Using FrontPage Database Connections

To create a database connection in FrontPage, you must enter information about the database, such as its name, ODBC driver, username, and password. FrontPage then writes this information to the global.asa file as a connection string.

However, connection strings must be constructed according to the ODBC driver being used, and the ODBC drivers are different on UNIX and Linux than on Windows. For this reason, before you can use Windows connections with Sun Chili!Soft ASP for UNIX or Linux, you must first edit them so they use the syntax described in "Creating Connection Strings" in this chapter.

In addition, when you use Microsoft SQL Server 6.5 or Microsoft Access databases with Sun Chili!Soft ASP for UNIX or Linux, you must use system DSNs in connection strings; you cannot use DSN-less connection strings or file DSNs. To use a system DSN for connecting to a particular database, your administrator must create a DSN for the database on the ASP Server. In addition, for SQL Server 6.5 and Access, your system administrator must configure SequeLink, as described in "Configuring SequeLink" in "Chapter 3: Managing Sun Chili!Soft ASP."

When you are using Sun Chili!Soft ASP for UNIX or Linux, consider migrating your Microsoft Access databases to dBASE as described in "Migrating an Access Database to dBASE" in this chapter. dBASE is relatively easy to learn and use and eliminates some of the platform compatibility problems you might otherwise experience.

For Microsoft SQL Server 7.0 or 2000 databases, in addition to system DSNs, you can also use DSN-less connection strings and file DSNs. You should verify that your connection strings follow the syntax described in "Creating Connection Strings" in this chapter.

For all other databases supported by Sun Chili!Soft ASP, you can use system DSNs, file DSNs, and DSN-less connection strings.

See also:

Using FrontPage Database Features in this chapter

Connecting to a Database in this chapter

Displaying Data on a Web Page with FrontPage

With Sun Chili!Soft ASP, FrontPage developers can continue to use FrontPage features for connecting to a database and displaying its information on an ASP page. Sun Chili!Soft ASP supports all methods for displaying database data that are generated by the FrontPage Database Results Wizard, including:

- Table format
- List format
- Drop-down menu format

By using the Database Results Wizard, developers can easily present the most recent data each time a user views and refreshes a page. Sun Chili!Soft ASP also supports the FrontPage "Send To Database" HTML form handler feature, and the **Recordset** navigation toolbar generated by the Database Results Wizard for moving quickly through the pages of records returned by a query.

Important

When using the FrontPage Database Results Wizard, you must first create the ASP pages locally on your workstation, and then publish them on the server running the ASP Server and FrontPage Server Extensions. After moving the ASP pages, you can later use FrontPage to edit them on the server. Note that you must change the connection strings created by FrontPage for them to work with Sun Chili!Soft ASP for UNIX or Linux. For more information, see "Using FrontPage Database Connections" in this section.

See also:

Using FrontPage Database Connections in this chapter

Connecting to a Database in this chapter

Migrating a Microsoft Access Database to dBASE

Microsoft Access databases are compatible with Sun Chili!Soft ASP running on Windows, but these databases do not run on UNIX or Linux systems. You have two options for connecting to a Microsoft Access database with Sun Chili!Soft ASP for UNIX or Linux.

First, you can use SequeLink. The necessary steps for creating the connection string are described in "Creating Connection Strings" in this chapter. The steps the system administrator must take are described in "Configuring SequeLink" in "Chapter 3: Managing Sun Chili!Soft ASP."

Alternatively, if you use FrontPage, you can easily migrate your Microsoft Access database to dBASE by using the Microsoft Access **Export Table** feature. You can then import the resulting folder of files to your FrontPage Web and use the Database Results Wizard. The dBASE database management system is relatively easy to learn and use.

If you have moved a dBASE-based Web application to UNIX and then find that Sun Chili!Soft ASP cannot open the database, make sure that the file extensions of your dBASE files are in all capital letters (that is, *.DBF).

Note

dBASE databases do not support multi-table joins on UNIX.

See also:

Connecting to a Database in this chapter

Using FrontPage Database Features in this chapter

Developing International Applications

By default, the Sun Chili!Soft ASP Server displays content in United States (US) English and uses US date, time, and currency formats. If you want to deliver ASP applications in locales and languages other than US English, your administrator can change the locale specified for the ASP Server, as described in "Configuring International Support" in "Chapter 3: Managing Sun Chili!Soft ASP." This ensures that the characters in the specified language display properly and that date, time, and currency formats are correct. Ask your administrator which locales are available.

Regardless of the locale for which your server is configured, you can dynamically change how certain content (such as date, time, and currency) is formatted so that it is appropriate for a given locale. You can do this from within an ASP page by changing the value of the **Session.LCID** property.

The following example shows how to display the current date first in German and then in English, using the **Session.LCID** property:

```
<%

Session.LCID = &H0407 ' specify Germany/German
```

```
Response.Write FormatDateTime( Date, vbLongDate ) & "<BR>" & vbNewLine

Session.LCID = &H0409 ' specify USA/English

Response.Write FormatDateTime( Date, vbLongDate ) & "<BR>" & vbNewLine

%>
```

Although you can dynamically change locales via **Session.LCID**, you can not effectively change code pages in this release of Sun Chili!Soft ASP by using **Session.CodePage**. This means that any characters that are not supported by the CODEPAGE specified for the ASP Server locale are not reproduced correctly. The major exception to this is characters falling within the normal ASCII range (0x00 to 0x7E), in which the graphical representations are the same in all languages for almost all characters. (The rare exceptions include the 0x5C character, which displays as a backslash in English but as a Yen symbol in Japanese.)

Notes about Japanese character support

Sun Chili!Soft ASP supports only the Shift-JIS encoding of Japanese characters and does not support "extended" or "user-defined" characters. Please note that this applies to all Japanese usage in ASP pages, including literal strings in the source files, text stored to files via the **Scripting.FileSystemObject**, text stored to databases via the various ADODB objects and methods, and so forth. (The implementation of ADO used with Sun Chili!Soft ASP is called ADODB.) Similarly, all output from ASP to browsers is in Shift-JIS only.

If a field is created as VarChar(nn) or Char(nn) then nn actually represents the number of bytes of data that can be stored in that field. Since the majority of Shift-JIS characters occupy two bytes of memory, fields should be specified with a size that is twice the maximum number of Shift-JIS characters that they need to hold.

Note about DB2 and locale

You must connect to a DB2 database that was created in the same locale in which the Web server and the Sun Chili!Soft ASP Server are running. If you do not, upon attempting to make the database connection from an ASP page, you might receive the following error message:

"There is no available conversion for the source code page "932" to the target code page "1252". Reason Code "1". SQLSTATE=57017"

To address this problem, create and connect to a database that is in the appropriate locale.

See also:
Understanding Code Pages in this chapter.

Understanding Code Pages

When you are building an ASP application that must support non-US-English users, the application must support character set conversions. Internally, ASP and the language engine it calls, VBScript or JScript, speak in Unicode strings. However, Web page content can be in ANSI, DBCS, or another character-encoding scheme. Therefore, when an HTTP request from a browser includes either form or query string values, they must be converted from the character set used by

the browser into Unicode for processing by an ASP script. These conversions map characters from one code page, which is a set of characters organized in some scheme, such as ANSI, to another. For example, the value that refers to the letter "a" in ANSI is converted to the different value that refers to that same letter "a" in Unicode. Similarly, when output is sent back to the browser, any strings returned by scripts must be converted from Unicode back to the code page used by the client.

These internal conversions are done using the default code page of the Web server. This works great if the users and the server are all speaking the same language (more precisely, if they use the same code page). However, for example, if you have a Japanese client hitting an English server, the code page translations do not work because ASP treats Japanese characters as if they are English.

See also:

Developing International Applications in this chapter.

Publishing a Sun Chili!Soft ASP Application

To publish an ASP application, you save the application files in the directory defined for that application on your Web server (be sure that the directory has either **Script** or **Execute** permission enabled). Your administrator can set up the ASP application directory on the server by using the procedure in "Adding an ASP Application" in "Chapter 3: Managing Sun Chili!Soft ASP." For more information about how ASP applications are structured, see "Creating the Basic ASP Application" in this chapter.

To verify that an ASP page is displaying properly, you can request the page with your browser by typing its URL. (Remember, ASP pages must be served, so you cannot request an *.asp file by typing its physical path.) After the page loads in your browser, you will notice that the server has returned an HTML page. This may seem strange at first, but remember that the ASP Server parses and executes all server-side scripts prior to sending the file. The user always receives standard HTML.

When publishing ASP pages created in FrontPage, be aware that if the **EnableParentPaths** configuration setting is **no**, the default, **CreateObject** ("**Scripting.FileSystemObject**") calls generated in the global.asa file by FrontPage will not work. Your system administrator must either change **EnableParentPaths** to **yes**, or else you must change the code that FrontPage generated in the global.asa file to **Server.CreateObject** ("**Scripting.FileSystemObject**"). For more information, see "Configuring File System Access" in "Chapter 3: Managing Sun Chili!Soft ASP."

Chapter 5: Developer's Reference

This chapter contains the following reference information for ASP developers:

- ADO Component Reference
- ASP Built-in Objects Reference
- ASP Component Reference
- Chili!Beans Component Reference
- Component Programmer's Reference
- JScript Language Reference
- SpicePack Component Reference
- VBScript Language Reference

ADO Component Reference

Sun Chili!Soft ASP includes ActiveX Data Objects (ADO) for connecting ASP applications to databases. ADO is a set of objects that provide a mechanism to access information from ODBC-compliant data sources.

This section provides the following ADO reference information:

- ADO Overview
- ADO Objects
- ADO Command Object
- ADO Connection Object
- ADO Error Object
- ADO Field Object
- ADO Parameter Object
- ADO Property Object
- ADO Recordset Object
- ADO Collections

ADO Overview

The implementation of ADO used with Sun Chili!Soft ASP is called ADODB. ADO enables client applications to access and manipulate data in a database server from a variety of different

vendors in the same manner. With ADO, data is updated and retrieved using a variety of existing methods (including SQL). In the context of ASP, using ADO typically involves writing script procedures that interact with a database and use HTML to display information from data sources.

In ADO, the **Recordset** object is the main interface to data. An example of the minimal VBScript code to generate a recordset from an ODBC data source is as follows:

```
set rstMain = CreateObject("ADODB.Recordset")
rstMain.Open "SELECT * FROM authors", _
"DATABASE=pubs;UID=sa;PWD=;DSN=Publishers"
```

This generates a forward-only, read-only **Recordset** object useful for scanning data. A slightly more functional recordset can be generated as follows:

```
set rstMain = CreateObject("ADODB.Recordset")
rstMain.Open "SELECT * FROM authors", _
"DATABASE=pubs;UID=sa;PWD=;DSN=Publishers",
adOpenKeyset, adLockOptimistic
```

This creates a fully scrollable and updateable recordset.

Note

Adovbs.inc & Adojavas.inc: For applications that use VBScript (for example, Active Server Pages), you must include the Adovbs.inc file in your code in order to call ADO constants by name (use Adojavas.inc for JScript). Always refer to constants by name rather than by value since the values may change from one version to the next.

Note

Updatable Cursor support: Microsoft and Sun Chili!Soft use the Positioned Update and Positioned SQL features of ODBC to implement the **AddNew**, **Update**, and **Delete** methods of the ADO Recordset Object . For some of the supplied ODBC drivers these features are not implemented at all (MySQL, PostgreSQL). For others the support is incomplete (DataDirect SequeLink driver). For these drivers, Sun Chili!Soft uses the implementation of updatable cursors in the ODBC Driver Manager to supply the missing functionality. This works well for recordsets whose fields contain string or numeric data as well as a primary key, auto-increment, or timestamp fields. However, in recordsets containing binary fields or recordsets with duplicate rows, updates, inserts and deletes should be done using the **Execute** method of the **Connection** object. **Connection.Execute** will execute any SQL statement recognized by the database regardless of the capabilities of the ODBC driver.

Note

Linux and multiple SELECT statements: On Linux, ADO does not support stored procedures with multiple SELECT statements.

In ADO, the object hierarchy is de-emphasized. Unlike Data Access Objects (DAO) or Remote Data Objects (RDO), you do not have to navigate through a hierarchy to create objects, because

most ADO objects can be independently created. This allows you to create and track only the objects you need. This model also results in fewer ADO objects, and thus a smaller working set.

ADO supports the following key features for building client/server and Web-based applications:

- Independently created objects.
- Support for stored procedures with in/out parameters and return values.
- Different cursor types, including the potential for support of back-end-specific cursors.
- Advanced recordsetcache management.
- Support for limits on number of returned rows and other query goals.

ADO Objects

ADO provides two objects for managing connections with data sources (**Connection** and **Command**), two objects for managing the data returned from a data source (**Field** and **Recordset**) and three secondary objects (**Parameters**, **Properties**, and **Errors**) for managing information about ADO.

Object	Description
ADO Command Object	Defines a specific command to execute against a data source.
ADO Connection Object	Represents an open connection to a data source.
ADO Error Object	Provides specific details about each ADO error.
ADO Field Object	Represents a column of data with a common data type.
ADO Parameter Object	Represents a parameter or argument associated with a **Command** object based on a parameterized query or stored procedure.
ADO Property Object	Represents a dynamic characteristic of an ADO object that is defined by the provider. *This object is not currently supported on UNIX.*
ADO Recordset Object	Represents the entire set of records from a database table or the results of an executed command.

ADO Command Object

ADO Command Object

The **Command** object defines a specific command to execute against a data source.

ADO Command Object Collections

ADO Parameters Collection	Contains all the **Parameter** objects of a **Command**

	object.
ADO Properties Collection	Contains all the **Property** objects for a specific instance of a **Command** object. *This collection is not currently supported on UNIX.*

ADO Command Object Methods

ADO Command Object CreateParameter Method	Creates a new **Parameter** object with the specified properties.
ADO Command Object Execute Method	Executes the query, SQL statement, or stored procedure specified in the **CommandText** property.

ADO Command Object Properties

ADO Command Object ActiveConnection Property	The **Connection** object to which the specified **Command** object currently belongs.
ADO Command Object CommandText Property	The text of a command that you want to issue against a provider.
ADO Command Object CommandTimeout Property	How long to wait while executing a command before terminating the command and issuing an error.
ADO Command Object CommandType Property	The type of **Command** object.
ADO Command Object Name Property	The name of a specific **Command** object. *This property is not currently supported on UNIX*
ADO Command Object Prepared Property	Whether or not to save a compiled version of a command before execution. *This property is not currently supported on UNIX.*
ADO Command Object State Property	The current state of the **Command** object. *This property is not currently supported on UNIX*

ADO Command Object Remarks

A **Command** object is used to query a database, return records in a ADO Recordset Object, execute bulk operations, or manipulate the structure of a database. It is a definition of a specific command that you intend to execute against a data source.

The collections, methods, and properties of a **Command** object are used to:

- Define the executable text of the command (for example, an SQL statement) with the ADO Command Object CommandText Property.
- Define parameterized queries or stored procedure arguments with ADO Parameter Object objects and the ADO Parameters Collection.
- Execute a command and return a ADO Recordset Object if appropriate with the ADO Command Object Execute Method.

- Specify the type of command with the ADO Command Object CommandType Property prior to execution to optimize performance.
- Set the number of seconds a provider will wait for a command to execute with the **CommandTimeout** property.
- Associate an open connection with a **Command** object by setting its property.
- Set the ADO Command Object Name Property to identify the **Command** object as a method on the associated ADO Connection Object.
- Pass a **Command** object to the ADO Recordset Object Source Property of an ADO Recordset Object in order to obtain data.

To execute a query without using a **Command** object, pass a query string to the ADO Connection Object Execute Method of an ADO Connection Object or to the ADO Recordset Object Open Method of an ADO Recordset Object. However, a **Command** object is required when you want to retain the command text and re-execute it, or use query parameters.

To create a **Command** object independently of a previously defined **Connection** object, set its **ActiveConnection** property to a valid connection string. ADO still creates a **Connection** object, but it doesn't assign that object to an object variable. However, if you are associating multiple **Command** objects with the same connection, you should explicitly create and open a **Connection** object; this assigns the **Connection** object to an object variable. If you do not set the **Command** object's **ActiveConnection** property to this object variable, ADO creates a new **Connection** object for each **Command** object, even if you use the same connection string.

To execute a **Command**, simply call it by its ADO Command Object Name Property on the associated **Connection** object. The **Command** must have its **ActiveConnection** property set to the **Connection** object. If the **Command** has parameters, pass values for them as arguments to the method.

Depending on the functionality of the provider, some **Command** collections, methods, or properties may generate an error when referenced.

ADO Command Object Methods

ADO Command Object CreateParameter Method

Creates a new **Parameter** object with the specified properties.

CreateParameter Method Syntax (ADO Command Object)

```
Set parameter = command.CreateParameter (
 Name, Type, Direction, Size, Value)
```

CreateParameter Method Parameters (ADO Command Object)

parameter

The new ADO Parameter Object.

Name

An optional **String** representing the name of the **Parameter** object.

Type

An optional **Long** value specifying the data type of the **Parameter** object. See the ADO Parameter Object Type Property for valid settings.

Direction

An optional **Long** value specifying the type of **Parameter** object. See the ADO Parameter Object Direction Property for valid settings.

Size

An optional **Long** value specifying the maximum length for the parameter value in characters or bytes.

Value

An optional **varValue** specifying the value for the **Parameter** object.

CreateParameter Method Return Value (ADO Command Object)

Returns a **Parameter** object.

CreateParameter Method Remarks (ADO Command Object)

Use the **CreateParameter** method to create a new ADO Parameter Object with the specified name, type, direction, size, and value. Any values you pass in the arguments are written to the corresponding **Parameter** properties.

This method does not automatically append the **Parameter** object to the ADO Parameters Collection of a **Command** object. This lets you set additional properties whose values ADO will validate when you append the **Parameter** object to the collection.

If you specify a variable-length data type in the *Type* argument, you must either pass a *Size* argument or set the ADO Parameter Object Size Property of the **Parameter** object before appending it to the **Parameters** collection; otherwise, an error occurs.

CreateParameter Method Examples (ADO Command Object)

See the ADO Collections Append Method example.

ADO Command Object Execute Method

Executes the query, SQL statement, or stored procedure specified in the **CommandText** property.

Object Execute Method Syntax (ADO Command Object)

For a row-returning Command:

```
Set recordset = command.Execute(
 RecordsAffected, Parameters, Options )
```

For a non-row-returning Command:

```
command.Execute RecordsAffected, Parameters, Options
```

Object Execute Method Parameters (ADO Command Object)

RecordsAffected

An optional **Long** variable to which the provider returns the number of records that the operation affected.

Parameters

An optional **Variant** array of parameter values passed with an SQL statement. (Output parameters will not return correct values when passed in this argument.)

Options

An optional **Long** value that indicates how the provider should evaluate the **CommandText** property of the **Command** object:

Constant	Description
adCmdText	The provider should evaluate **CommandText** as a textual definition of a command, such as a SQL statement.
adCmdTable	The provider should evaluate **CommandText** as a table name.
adCmdStoredProc	The provider should evaluate **CommandText** as a stored procedure.
adCmdUnknown	The type of command in **CommandText** is not known.

See the ADO Command Object CommandType Property for a more detailed explanation of the four constants in this list.

Object Execute Method Remarks (ADO Command Object)

Using the **Execute** method on a **Command** object executes the query specified in the **CommandText** property of the object. If the **CommandText** property specifies a row-returning query, any results the execution generates are stored in a new ADO Recordset Object. If the command is not a row-returning query, the provider returns a closed **Recordset** object. Some application languages allow you to ignore this return value if no recordset is desired.

If the query has parameters, the current values for the **Command** object's parameters are used unless you override these with parameter values passed with the **Execute** call. You can override a subset of the parameters by omitting new values for some of the parameters when calling the **Execute** method. The order in which you specify the parameters is the same order in which the method passes them. For example, if there were four (or more) parameters and you wanted to pass new values for only the first and fourth parameters, you would pass Array(var1,,,var4) as the *Parameters* argument.

Note

Output parameters will not return correct values when passed in the *Parameters* argument.

Object Execute Method Return Values (ADO Command Object)

Returns a **Recordset** object reference.

Object Execute Method Examples (ADO Command Object)

This VBScript example demonstrates the **Execute** method when run from both a **Command** object and an ADO Connection Object. It also uses the ADO Recordset Object Requery Method to retrieve current data in a recordset, and the ADO Collections Clear Method to clear the contents of the ADO Errors Collection. The ExecuteCommand and PrintOutput procedures are required for this procedure to run.

```
<!-- #Include file="ADOVBS.INC" -->

<HTML><HEAD>

<TITLE>ADO 1.5 Execute Method</TITLE></HEAD>

<BODY>

<FONT FACE="MS SANS SERIF" SIZE=2>

<Center><H3>ADO Execute Method</H3><H4>Recordset Retrieved Using
Connection Object</H4>

<TABLE WIDTH=600 BORDER=0>

<TD VALIGN=TOP ALIGN=LEFT COLSPAN=3><FONT SIZE=2>

<!--- Recordsets retrieved using Execute method of Connection and
Command Objects-->

<%

Set OBJdbConnection = Server.CreateObject("ADODB.Connection")

OBJdbConnection.Open "AdvWorks"

SQLQuery = "SELECT * FROM Customers"

'First Recordset RSCustomerList

Set RSCustomerList = OBJdbConnection.Execute(SQLQuery)

Set OBJdbCommand = Server.CreateObject("ADODB.Command")

OBJdbCommand.ActiveConnection = OBJdbConnection

SQLQuery2 = "SELECT * From Products"

OBJdbCommand.CommandText = SQLQuery2

Set RsProductList = OBJdbCommand.Execute

%>

<TABLE COLSPAN=8 CELLPADDING=5 BORDER=0>

<!-- BEGIN column header row for Customer Table-->

<TR><TD ALIGN=CENTER BGCOLOR="#008080">
```

```
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>Company
Name</FONT>

</TD>

<TD ALIGN=CENTER BGCOLOR="#008080">

<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>Contact
Name</FONT>

</TD>

<TD ALIGN=CENTER WIDTH=150 BGCOLOR="#008080">

<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>E-mail
address</FONT>

</TD>

<TD ALIGN=CENTER BGCOLOR="#008080">

<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>City</FONT>

</TD>

<TD ALIGN=CENTER BGCOLOR="#008080">

<FONT STYLE="ARIAL NARROW" COLOR="#ffffff"
SIZE=1>State/Province</FONT>

</TD></TR>

<!--Display ADO Data from Customer Table-->

<% Do While Not RScustomerList.EOF %>

<TR>

<TD BGCOLOR="f7efde" ALIGN=CENTER>

<FONT STYLE="ARIAL NARROW" SIZE=1>

<%= RSCustomerList("CompanyName")%>

</FONT></TD>

<TD BGCOLOR="f7efde" ALIGN=CENTER>

<FONT STYLE="ARIAL NARROW" SIZE=1>

<%= RScustomerList("ContactLastName") & ", " %>

<%= RScustomerList("ContactFirstName") %>

</FONT></TD>

<TD BGCOLOR="f7efde" ALIGN=CENTER>

<FONT STYLE="ARIAL NARROW" SIZE=1>

<%= RScustomerList("ContactLastName")%>
```

```
</FONT></TD>
<TD BGCOLOR="f7efde" ALIGN=CENTER>
<FONT STYLE="ARIAL NARROW" SIZE=1>
<%= RScustomerList("City")%>
</FONT></TD>
<TD BGCOLOR="f7efde" ALIGN=CENTER>
<FONT STYLE="ARIAL NARROW" SIZE=1>
<%= RScustomerList("StateOrProvince")%>
</FONT></TD>
</TR>
<!-Next Row = Record Loop and add to html table-->
<%
RScustomerList.MoveNext
Loop
RScustomerList.Close
%>
</TABLE><HR>
<H4>Recordset Retrieved Using Command Object</H4>
<TABLE COLSPAN=8 CELLPADDING=5 BORDER=0>
<!-- BEGIN column header row for Product List Table-->
<TR><TD ALIGN=CENTER BGCOLOR="#800000">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>Product
Type</FONT>
</TD>
<TD ALIGN=CENTER BGCOLOR="#800000">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>Product
Name</FONT>
</TD>
<TD ALIGN=CENTER WIDTH=350 BGCOLOR="#800000">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>Product
Description</FONT>
</TD>
<TD ALIGN=CENTER BGCOLOR="#800000">
```

```
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>Unit Price</FONT>
</TD></TR>
<!-- Display ADO Data Product List-->
<% Do While Not RsProductList.EOF %>
<TR>
<TD BGCOLOR="f7efde" ALIGN=CENTER>
<FONT STYLE="ARIAL NARROW" SIZE=1>
<%= RsProductList("ProductType")%>
</FONT></TD>
<TD BGCOLOR="f7efde" ALIGN=CENTER>
<FONT STYLE="ARIAL NARROW" SIZE=1>
<%= RsProductList("ProductName")%>
</FONT></TD>
<TD BGCOLOR="f7efde" ALIGN=CENTER>
<FONT STYLE="ARIAL NARROW" SIZE=1>
<%= RsProductList("ProductDescription")%>
</FONT></TD>
<TD BGCOLOR="f7efde" ALIGN=CENTER>
<FONT STYLE="ARIAL NARROW" SIZE=1>
<%= RsProductList("UnitPrice")%>
</FONT></TD>
<!-- Next Row = Record -->
<%
RsProductList.MoveNext
Loop
'Remove objects from memory to free resources
RsProductList.Close
OBJdbConnection.Close
Set ObJdbCommand = Nothing
Set RsProductList = Nothing
Set OBJdbConnection = Nothing
%>
```

```
</TABLE></FONT></Center></BODY></HTML>
```

ADO Command Object Properties

ADO Command Object ActiveConnection Property

Specifies to which **Connection** object the specified **Command** object currently belongs.

ActiveConnection Property Return Values (ADO Command Object)

Sets or returns a **String** containing the definition for a connection or a **Connection** object. Default is a **Null** object reference.

ActiveConnection Property Remarks (ADO Command Object)

Use the **ActiveConnection** property to determine the **Connection** object over which the specified **Command** object will execute.

For **Command** objects, the **ActiveConnection** property is read/write. If you attempt to call the ADO Command Object Execute Method on a **Command** object before setting this property to an open ADO Connection Object or valid connection string, an error occurs. Setting the **ActiveConnection** property to *Nothing* disassociates the **Command** object from the current **Connection** and causes the provider to release any associated resources on the data source. You can then associate the **Command** object with the same or another **Connection** object. Some providers allow you to change the property setting from one **Connection** to another, without having to first set the property to *Nothing*.

If the ADO Parameters Collection of the **Command** object contains parameters supplied by the provider, the collection is cleared if you set the **ActiveConnection** property to *Nothing* or to another **Connection** object. If you manually create ADO Parameter Object objects and use them to fill the **Parameters** collection of the **Command** object, setting the **ActiveConnection** property to *Nothing* or to another **Connection** object leaves the **Parameters** collection intact.

Closing the **Connection** object with which a **Command** object is associated sets the **ActiveConnection** property to *Nothing*. Setting this property to a closed **Connection** object generates an error.

ActiveConnection Property Example (ADO Command Object)

This Visual Basic example uses the **ActiveConnection, ADO Command Object CommandText Property, CommandTimeout,** ADO Command Object CommandType Property**,** ADO Field Object ActualSize Property**,** and ADO Parameter Object Direction Property properties to execute a stored procedure:

```
Public Sub ActiveConnectionX()

Dim cnn1 As ADODB.Connection

Dim cmdByRoyalty As ADODB.Command

Dim prmByRoyalty As ADODB.Parameter

Dim rstByRoyalty As ADODB.Recordset
```

```
Dim rstAuthors As ADODB.Recordset
Dim intRoyalty As Integer
Dim strAuthorID As String
Dim strCnn As String
` Define a command object for a stored procedure.
Set cnn1 = New ADODB.Connection
strCnn = "driver={SQL Server};server=srv;" & _
"uid=sa;pwd=;database=pubs"
cnn1.Open strCnn
Set cmdByRoyalty = New ADODB.Command
Set cmdByRoyalty.ActiveConnection = cnn1
cmdByRoyalty.CommandText = "byroyalty"
cmdByRoyalty.CommandType = adCmdStoredProc
cmdByRoyalty.CommandTimeout = 15
` Define the stored procedure's input parameter.
intRoyalty = Trim(InputBox( _
"Enter royalty:"))
Set prmByRoyalty = New ADODB.Parameter
prmByRoyalty.Type = adInteger
prmByRoyalty.Size = 3
prmByRoyalty.Direction = adParamInput
prmByRoyalty.Value = intRoyalty
cmdByRoyalty.Parameters.Append prmByRoyalty
` Create a recordset by executing the command.
Set rstByRoyalty = cmdByRoyalty.Execute()
` Open the Authors table to get author names for display.
Set rstAuthors = New ADODB.Recordset
rstAuthors.Open "authors", strCnn, , , adCmdTable
` Print current data in the recordset, adding
` author names from Authors table.
Debug.Print "Authors with " & intRoyalty & _
" percent royalty"
```

```
Do While Not rstByRoyalty.EOF
strAuthorID = rstByRoyalty!au_id
Debug.Print , rstByRoyalty!au_id & ", ";
rstAuthors.Filter = "au_id = '" & strAuthorID & "'"
Debug.Print rstAuthors!au_fname & " " & _
rstAuthors!au_lname
rstByRoyalty.MoveNext
Loop
rstByRoyalty.Close
rstAuthors.Close
cnn1.Close
End Sub
```

ADO Command Object CommandText Property

Contains the text of a command that you want to issue against a provider.

CommandText Property Return Values

Sets or returns a **String** value containing a provider command, such as an SQL statement, a table name, or a stored procedure call. Default is "" (zero-length string).

CommandText Property Remarks

Use the **CommandText** property to set or return the text of a **Command** object. Usually, this will be an SQL statement, but can also be any other type of command statement recognized by the provider, such as a stored procedure call. An SQL statement must be of the particular dialect or version supported by the provider's query processor.

If the ADO Command Object Prepared Property of the **Command** object is set to **True** and the **Command** object is bound to an open connection when you set the **CommandText** property, ADO prepares the query (that is, a compiled form of the query is stored by the provider) when you call the ADO Command Object Execute Method or ADO Connection Object Open Method methods. The **Prepared** property is not currently supported on UNIX.

Depending on the ADO Command Object CommandType Property setting, ADO may alter the **CommandText** property. You can read the **CommandText** property at any time to see the actual command text that ADO will use during execution.

CommandText Property Example

See the **ActiveConnection** property.

ADO Command Object CommandTimeout Property

How long to wait while executing a command before terminating the attempt and generating an error.

CommandTimeout Property Return Values (ADO Command Object)

Sets or returns a **Long** value that specifies, in seconds, how long to wait for a command to execute. Default is 30.

CommandTimeout Property Remarks (ADO Command Object)

Use the **CommandTimeout** property on a **Command** object to allow the cancellation of an ADO Command Object Execute Method call due to delays from network traffic or heavy server use. If the interval set in the **CommandTimeout** property elapses before the command completes execution, an error occurs and ADO cancels the command. If you set the property to zero, ADO will wait indefinitely until the execution is complete. Make sure the provider and data source to which you are writing code supports the **CommandTimeout** functionality.

The **CommandTimeout** setting on a **Connection** object has no effect on the **CommandTimeout** setting on a **Command** object on the same **Connection**; that is, the **Command** object's **CommandTimeout** property does not inherit the value of the **Connection** object's **CommandTimeout** value.

CommandTimeout Property Examples (ADO Command Object)

See the **ActiveConnection** property.

ADO Command Object CommandType Property

The type of a **Command** object.

CommandType Property Return Values (ADO Command Object)

Sets or returns one of the following **CommandTypeEnum** values:

Constant	Description
adCmdText	Evaluates **CommandText** as a textual definition of a command.
adCmdTable	Evaluates **CommandText** as a table name.
adCmdStoredProc	Evaluates **CommandText** as a stored procedure.
adCmdUnknown	(Default) The type of command in the **CommandText** property is not known.

CommandType Property Remarks (ADO Command Object)

Use the **CommandType** property to optimize evaluation of the ADO Command Object CommandText Property.

If the **CommandType** property value equals **adCmdUnknown** (the default value), you may experience diminished performance because ADO must make calls to the provider to determine if the **CommandText** property is an SQL statement, a stored procedure, or a table name. If you know what type of command you're using, setting the **CommandType** property instructs ADO to

go directly to the relevant code. If the **CommandType** property does not match the type of command in the **CommandText** property, an error occurs when you call the ADO Command Object Execute Method.

CommandType Property Example (ADO Command Object)

See the **ActiveConnection** property.

ADO Command Object Name Property

The name of an object. *This property is not currently supported on UNIX*

Name Property Return Values (ADO Command Object)

Sets or returns a **String** value. The value is read/write.

Name Property Remarks (ADO Command Object)

Use the **Name** property to assign a name to or retrieve the name of a **Command** object.

ADO Command Object Prepared Property

Determines whether or not the provider saves a compiled version of a command before execution. *This property is not currently supported on UNIX.*

Prepared Property Return Values

Sets or returns a **Boolean** value.

Prepared Property Remarks

Use the **Prepared** property to have the provider save a prepared (or compiled) version of the query specified in the ADO Command Object CommandText Property before a **Command** object's first execution. This may slow a command's first execution, but once the provider compiles a command, the provider will use the compiled version of the command for any subsequent executions, which will result in improved performance.

If the property is **False**, the provider will execute the **Command** object directly without creating a compiled version.

If the provider does not support command preparation, it may return an error as soon as this property is set to **True**. If it does not return an error, it simply ignores the request to prepare the command and sets the **Prepared** property to **False**.

Prepared Property Example

This Visual Basic example demonstrates the **Prepared** property by opening two **Command** objects: one prepared and one not prepared.

```
Public Sub PreparedX()

Dim cnn1 As ADODB.Connection

Dim cmd1 As ADODB.Command
```

```
Dim cmd2 As ADODB.Command
Dim strCnn As String
Dim strCmd As String
Dim sngStart As Single
Dim sngEnd As Single
Dim sngNotPrepared As Single
Dim sngPrepared As Single
Dim intLoop As Integer
` Open a connection.
strCnn = "driver={SQL Server};server=srv;" & _
"uid=sa;pwd=;database=pubs"
Set cnn1 = New ADODB.Connection
cnn1.Open strCnn
` Create two command objects for the same
` command -- one prepared and one not prepared.
strCmd = "SELECT title, type FROM titles ORDER BY type"
Set cmd1 = New ADODB.Command
Set cmd1.ActiveConnection = cnn1
cmd1.CommandText = strCmd
Set cmd2 = New ADODB.Command
Set cmd2.ActiveConnection = cnn1
cmd2.CommandText = strCmd
cmd2.Prepared = True
` Set timer, then execute unprepared command 20 times.
sngStart = Timer
For intLoop = 1 To 20
cmd1.Execute
Next intLoop
sngEnd = Timer
sngNotPrepared = sngEnd - sngStart
` Reset the timer, then execute the prepared
` command 20 times.
```

```
sngStart = Timer
For intLoop = 1 To 20
cmd2.Execute
Next intLoop
sngEnd = Timer
sngPrepared = sngEnd - sngStart
` Display performance results.
MsgBox "Performance Results:" & vbCr & _
" Not Prepared: " & Format(sngNotPrepared, _
"##0.000") & " seconds" & vbCr & _
" Prepared: " & Format(sngPrepared, _
"##0.000") & " seconds"
cnn1.Close
End Sub
```

ADO Command Object State Property

Describes the current state of an object. *This property is not currently supported on UNIX*

State Property Return Values (ADO Command Object)

Sets or returns a **Long** value that can be one of the following constants:

Constant	Description
adStateClosed	The object is closed. Default.
adStateOpen	The object is open.

State Property Remarks (ADO Command Object)

You can use the **State** property to determine the current state of a given object at any time.

ADO Connection Object

ADO Connection Object

A **Connection** object represents an open connection to a data source.

ADO Connection Object Collections

ADO Errors Collection	Contains all stored **Error** objects that pertain to an ADO operation.
ADO Properties Collection	All **Property** objects for a specific instance of a **Connection**

	object. *This collection is not currently supported on UNIX.*

ADO Connection Object Methods

ADO Connection Object BeginTrans, CommitTrans, and RollbackTrans Methods	Begins a new database transaction within a **Connection** object.
ADO Connection Object Close Method	Closes an open **Connection** object and any dependent objects.
ADO Connection Object BeginTrans, CommitTrans, and RollbackTrans Methods	Saves any pending changes and ends the current transaction. It may also start a new transaction.
ADO Connection Object Execute Method	Executes the specified query, SQL statement, stored procedure, or provider-specified text.
ADO Connection Object Open Method	Opens a connection to a data source.
ADO Connection Object OpenSchema Method	Obtains database schema information from the provider. *This method is not currently supported on UNIX.*
ADO Connection Object BeginTrans, CommitTrans, and RollbackTrans Methods	Cancels any changes made during the current transaction and ends the transaction. It may also start a new transaction.

ADO Connection Object Properties

ADO Connection Object Attributes Property	One or more characteristics of an object.
ADO Connection Object CommandTimeout Property	How long to wait while executing a command before terminating the command and issuing and error.
ADO Connection Object ConnectionString Property	Contains the information used to establish a connection to a data source.
ADO Connection Object ConnectionTimeout Property	How long to wait while establishing a connection before terminating the attempt and issuing and error.
ADO Connection Object CursorLocation Property	The location of the cursor engine in a recordset.
ADO Connection Object DefaultDatabase Property	The default database for the **Connection** object.
ADO Connection Object IsolationLevel Property	The level of isolation for the **Connection** object.
ADO Connection Object Mode Property	The available permissions for modifying data in a **Connection** object.
ADO Connection Object	The name of a provider for a **Connection** object. *This property*

Provider Property	*is not available on UNIX.*
ADO Connection Object State Property	Describes the current state of the **Connection** object.
ADO Connection Object Version Property	The ADO version number.

ADO Connection Object Remarks

A **Connection** object represents a session with a data source. In the case of a client/server database system, it may represent an actual network connection to the server. Depending on the functionality of the provider, some collections, properties, and methods of the **Connection** object may not be available.

Use the collections, methods, and properties of a **Connection** object for:

- configuring the connection before opening it with the **ConnectionString, CommandTimeout,** and ADO Connection Object Mode Property properties.
- setting the **CursorLocation** property to invoke the Client Cursor Provider, which supports batch updates. *Batch updates are not currently supported on UNIX.*
- setting the default database for the connection with the **DefaultDatabase** property.
- setting the level of isolation for the transactions opened on the connection with the **IsolationLevel** property. *Transactions are not currently supported on UNIX.*
- specifying an OLE DB provider with the ADO Connection Object Provider Property.
- establishing and breaking the physical connection to the data source with the ADO Connection Object Open Method and ADO Connection Object Close Method methods.
- executing a command on the connection with the ADO Connection Object Execute Method and configuring the execution with the **CommandTimeout** property.
- managing transactions on the open connection, including nested transactions if the provider supports them, with the **BeginTrans, CommitTrans,** and **RollbackTrans** methods and the ADO Connection Object Attributes Property. *The transaction methods are not currently supported on UNIX.*
- examining errors returned from the data source with the ADO Errors Collection.
- reading the version from the ADO implementation in use with the ADO Connection Object Version Property.
- obtaining schema information about your database with the ADO Connection Object OpenSchema Method.

Note

To execute a query without using a **Command** object, pass a query string to the **Execute** method of a **Connection** object. However, a **Command** object is required when you want to retain the command text and re-execute it, or use query parameters.

ADO Connection Object Methods

ADO Connection Object Close Method

Closes an open object and any dependent objects.

Close Method Syntax (ADO Connection Object)

```
object.Close
```

Close Method Remarks (ADO Connection Object)

Use the **Close** method to close a **Connection** object to free any associated system resources. Closing an object does not remove it from memory; you may change its property settings and open it again later. To completely eliminate an object from memory, set the object variable to *Nothing*.

Using the **Close** method to close a **Connection** object also closes any active **Recordset** objects associated with the connection. An ADO Command Object associated with the **Connection** object you are closing will persist, but it will no longer be associated with a **Connection** object; that is, its **ActiveConnection** property will be set to *Nothing*. Also, the **Command** object's ADO Parameters Collection will be cleared of any provider-defined parameters.

You can later call the ADO Connection Object Open Method to reestablish the connection to the same or another data source. While the **Connection** object is closed, calling any methods that require an open connection to the data source generates an error.

Closing a **Connection** object while there are open ADO Recordset Object objects on the connection rolls back any pending changes in all of the **Recordset** objects. Explicitly closing a **Connection** object (calling the **Close** method) while a transaction is in progress generates an error. If a **Connection** object falls out of scope while a transaction is in progress, ADO automatically rolls back the transaction.

Close Method Examples (ADO Connection Object)

This VBScript example uses the **Open** and **Close** methods on both **Recordset** and **Connection** objects that have been opened.

```
<!-- #Include file="ADOVBS.INC" -->

<HTML><HEAD>

<TITLE>ADO 1.5 Open Method</TITLE>

</HEAD><BODY>

<FONT FACE="MS SANS SERIF" SIZE=2>

<Center><H3>ADO Open Method</H3>

<TABLE WIDTH=600 BORDER=0>

<TD VALIGN=TOP ALIGN=LEFT COLSPAN=3><FONT SIZE=2>

<!--- ADO Connection used to create 2 recordsets-->
```

```
<%
Set OBJdbConnection = Server.CreateObject("ADODB.Connection")
OBJdbConnection.Open "AdvWorks"
SQLQuery = "SELECT * FROM Customers"
'First Recordset RSCustomerList
Set RSCustomerList = OBJdbConnection.Execute(SQLQuery)
'Second Recordset RsProductist
Set RsProductList = Server.CreateObject("ADODB.Recordset")
RsProductList.CursorType = adOpenDynamic
RsProductList.LockType = adLockOptimistic
RsProductList.Open "Products", OBJdbConnection
%>
<TABLE COLSPAN=8 CELLPADDING=5 BORDER=0>
<!-- BEGIN column header row for Customer Table-->
<TR><TD ALIGN=CENTER BGCOLOR="#008080">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>Company
Name</FONT></TD>
<TD ALIGN=CENTER BGCOLOR="#008080">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>Contact
Name</FONT></TD>
<TD ALIGN=CENTER WIDTH=150 BGCOLOR="#008080">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>E-mail
address</FONT></TD>
<TD ALIGN=CENTER BGCOLOR="#008080">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>City</FONT></TD>
<TD ALIGN=CENTER BGCOLOR="#008080">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff"
SIZE=1>State/Province</FONT></TD></TR>
<!--Display ADO Data from Customer Table-->
<% Do While Not RScustomerList.EOF %>
<TR><TD BGCOLOR="f7efde" ALIGN=CENTER>
<FONT STYLE="ARIAL NARROW" SIZE=1>
<%= RSCustomerList("CompanyName")%>
```

```
</FONT></TD>
<TD BGCOLOR="f7efde" ALIGN=CENTER>
<FONT STYLE="ARIAL NARROW" SIZE=1>
<%= RScustomerList("ContactLastName") & ", " %>
<%= RScustomerList("ContactFirstName") %>
</FONT></TD>
<TD BGCOLOR="f7efde" ALIGN=CENTER>
<FONT STYLE="ARIAL NARROW" SIZE=1>
<%= RScustomerList("ContactLastName")%>
</FONT></TD>
<TD BGCOLOR="f7efde" ALIGN=CENTER>
<FONT STYLE="ARIAL NARROW" SIZE=1>
<%= RScustomerList("City")%>
</FONT></TD>
<TD BGCOLOR="f7efde" ALIGN=CENTER>
<FONT STYLE="ARIAL NARROW" SIZE=1>
<%= RScustomerList("StateOrProvince")%>
</FONT></TD></TR>
<!-Next Row = Record Loop and add to html table-->
<%
RScustomerList.MoveNext
Loop
RScustomerList.Close
OBJdbConnection.Close
%>
</TABLE>
<HR>
<TABLE COLSPAN=8 CELLPADDING=5 BORDER=0>
<!-- BEGIN column header row for Product List Table-->
<TR><TD ALIGN=CENTER BGCOLOR="#800000">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>Product
Type</FONT></TD>
<TD ALIGN=CENTER BGCOLOR="#800000">
```

```
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>Product
Name</FONT></TD>

<TD ALIGN=CENTER WIDTH=350 BGCOLOR="#800000">

<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>Product
Description</FONT></TD>

<TD ALIGN=CENTER BGCOLOR="#800000">

<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>Unit
Price</FONT></TD></TR>

<!-- Display ADO Data Product List-->

<% Do While Not RsProductList.EOF %>

<TR> <TD BGCOLOR="f7efde" ALIGN=CENTER>

<FONT STYLE="ARIAL NARROW" SIZE=1>

<%= RsProductList("ProductType")%>

</FONT></TD>

<TD BGCOLOR="f7efde" ALIGN=CENTER>

<FONT STYLE="ARIAL NARROW" SIZE=1>

<%= RsProductList("ProductName")%>

</FONT></TD>

<TD BGCOLOR="f7efde" ALIGN=CENTER>

<FONT STYLE="ARIAL NARROW" SIZE=1>

<%= RsProductList("ProductDescription")%>

</FONT></TD>

<TD BGCOLOR="f7efde" ALIGN=CENTER>

<FONT STYLE="ARIAL NARROW" SIZE=1>

<%= RsProductList("UnitPrice")%>

</FONT></TD>

<!-- Next Row = Record -->

<%

RsProductList.MoveNext

Loop

'Remove Objects from Memory Freeing

Set RsProductList = Nothing

Set OBJdbConnection = Nothing
```

```
%>
</TABLE></FONT></Center></BODY></HTML>
```

ADO Connection Object Execute Method

Executes the specified query, SQL statement, stored procedure, or provider-specific text.

Execute Method Syntax (ADO Connection Object)

For a non-row-returning command string:

```
connection.Execute CommandText, RecordsAffected, Options
```

For a row-returning command string:

```
Set recordset = connection.Execute (
 CommandText, RecordsAffected, Options)
```

Execute Method Parameters (ADO Connection Object)

CommandText

A **String** containing the SQL statement, table name, stored procedure, or provider-specific text to execute.

RecordsAffected

An optional **Long** variable to which the provider returns the number of records that the operation affected.

Options

An optional **Long** value that indicates how the provider should evaluate the **CommandText** argument:

Constant	Description
adCmdText	The provider should evaluate *CommandText* as a textual definition of a command.
adCmdTable	The provider should evaluate *CommandText* as a table name.
adCmdStoredProc	The provider should evaluate *CommandText* as a stored procedure.
adCmdUnknown	The type of command in the *CommandText* argument is not known.

See the ADO Command Object CommandType Property for a more detailed explanation of the four constants in this list.

Execute Method Return Values (ADO Connection Object)

Returns an ADO Recordset Object reference.

Execute Method Remarks (ADO Connection Object)

Using the **Execute** method on a **Connection** object executes whatever query you pass to the method in the *CommandText* argument on the specified connection. If the *CommandText*

argument specifies a row-returning query, any results the execution generates are stored in a new **Recordset** object. If the command is not a row-returning query, the provider returns a closed **Recordset** object.

The returned **Recordset** object is always a read-only, forward-only cursor. If you need a **Recordset** object with more functionality, first create a **Recordset** object with the desired property settings, then use the **Recordset** object's ADO Recordset Object Open Method to execute the query and return the desired cursor type.

The contents of the *CommandText* argument are specific to the provider and can be standard SQL syntax or any special command format that the provider supports.

Execute Method Examples (ADO Connection Object)

See the Command ADO Command Object Execute Method.

ADO Connection Object OpenSchema Method

Obtains database schema information from the provider.

OpenSchema Method Syntax

```
Set recordset = connection.OpenSchema (QueryType,
 Criteria, SchemaID)
```

OpenSchema Method Parameters

QueryType

The type of schema query to run. Can be any of the constants listed below.

Criteria

Optional array of query constraints for each **QueryType** option, as listed below:

QueryType values	Criteria values
adSchemaAsserts	CONSTRAINT_CATALOG CONSTRAINT_SCHEMA CONSTRAINT_NAME
adSchemaCatalogs	CATALOG_NAME
asSchemaCharacterSets	CHARACTER_SET_CATALOG CHARACTER_SET_SCHEMA CHARACTER_SET_NAME
adSchemaCheckConstraints	CONSTRAINT_CATALOG CONSTRAINT_SCHEMA CONSTRAINT_NAME
adSchemaCollations	COLLATION_CATALOG COLLATION_SCHEMA COLLATION_NAME

adSchemaColumnDomainUsage	DOMAIN_CATALOG DOMAIN_SCHEMA DOMAIN_NAME COLUMN_NAME
adSchemaColumnPrivileges	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME COLUMN_NAME GRANTOR GRANTEE
adSchemaColumns	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME
adSchemaConstraintTableUsage	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME COLUMN_NAME
adSchemaForeignKeys	PK_TABLE_CATALOG PK_TABLE_SCHEMA PK_TABLE_NAME FK_TABLE_CATALOG FK_TABLE_SCHEMA FK_TABLE_NAME
adSchemaIndexes	TABLE_CATALOG TABLE_SCHEMA INDEX_NAME TYPE TABLE_NAME
adSchemaKeyColumnUsage	CONSTRAINT_CATALOG CONSTRAINT_SCHEMA CONSTRAINT_NAME TABLE_CATALOG TABLE_SCHEMA TABLE_NAME COLUMN_NAME
adSchemaPrimaryKeys	PK_TABLE_CATALOG PK_TABLE_SCHEMA PK_TABLE_NAME
adSchemaProcedureColumns	PROCEDURE_CATALOG PROCEDURE_SCHEMA PROCEDURE_NAME COLUMN_NAME

adSchemaProcedures	PROCEDURE_CATALOG PROCEDURE_SCHEMA PROCEDURE_NAME PARAMETER_TYPE
adSchemaProviderSpecific	see Remarks
adSchemaProviderTypes	DATA_TYPE BEST_MATCH
adSchemaReferentialConstraints	CONSTRAINT_CATALOG CONSTRAINT_SCHEMA CONSTRAINT_NAME
adSchemaSchemata	CATALOG_NAME SCHEMA_NAME SCHEMA_OWNER
adSchemaSQLLanguages	<none>
adSchemaStatistics	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME
adSchemaTableConstraints	CONSTRAINT_CATALOG CONSTRAINT_SCHEMA CONSTRAINT_NAME TABLE_CATALOG TABLE_SCHEMA TABLE_NAME CONSTRAINT_TYPE
adSchemaTablePrivileges	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME
adSchemaTables	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME TABLE_TYPE
adSchemaTranslations	TRANSLATION_CATALOG TRANSLATION_SCHEMA TRANSLATION_NAME
adSchemaUsagePrivileges	OBJECT_CATALOG OBJECT_SCHEMA OBJECT_NAME OBJECT_TYPE GRANTOR GRANTEE

adSchemaViewColumnUsage	VIEW_CATALOG VIEW_SCHEMA VIEW_NAME
adSchemaViewTableUsage	VIEW_CATALOG VIEW_SCHEMA VIEW_NAME
adSchemaViews	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME

SchemaID

The GUID for a provider-schema schema query is not defined by the OLE DB 1.1 specification. This parameter is required if *QueryType* is set to **adSchemaProviderSpecific**; otherwise, it is not used.

OpenSchema Method Return Values

Returns an ADO Recordset Object that contains schema information.

OpenSchema Method Remarks

The **OpenSchema** method returns information about the data source, such as information about the tables on the server and the columns in the tables.

The *Criteria* argument is an array of values that can be used to limit the results of a schema query. Each schema query has a different set of parameters that it supports. The actual schemas are defined by the OLE DB specification under the "IDBSchemaRowset" interface. The ones supported in ADO 1.5 are listed above.

The constant **adSchemaProviderSpecific** is used for the *QueryType* argument if the provider defines its own non-standard schema queries outside those listed above. When this constant is used, the *SchemaID* argument is required to pass the GUID of the schema query to execute. If *QueryType* is set to **adSchemaProviderSpecific** but *SchemaID* is not provided, an error will result.

Providers are not required to support all of the OLE DB standard schema queries. Specifically, only **adSchemaTables**, **adSchemaColumns** and **adSchemaProviderTypes** are required by the OLE DB specification. However, the provider is not required to support the *Criteria* constraints listed above for those schema queries.

OpenSchema Method Example

This Visual Basic example uses the **OpenSchema** method to display the name and type of each table in the ***Pubs*** database.

```
Public Sub OpenSchemaX()

Dim cnn1 As ADODB.Connection

Dim rstSchema As ADODB.Recordset

Dim strCnn As String
```

```
Set cnn1 = New ADODB.Connection
strCnn = "driver={SQL Server};server=srv;" & _
"uid=sa;pwd=;database=pubs"
cnn1.Open strCnn
Set rstSchema = cnn1.OpenSchema(adSchemaTables)
Do Until rstSchema.EOF
Debug.Print "Table name: " & _
rstSchema!TABLE_NAME & vbCr & _
"Table type: " & rstSchema!TABLE_TYPE & vbCr
rstSchema.MoveNext
Loop
rstSchema.Close
cnn1.Close
End Sub
```

ADO Connection Object Open Method

Opens a connection to a data source.

Open Method Syntax (ADO Connection Object)

```
connection.Open ConnectionString, UserID, Password
```

Open Method Parameters (ADO Connection Object)

ConnectionString

An optional *String* containing connection information. See the **ConnectionString** property for details on valid settings.

UserID

An optional *String* containing a user name to use when establishing the connection.

Password

An optional *String* containing a password to use when establishing the connection.

Open Method Remarks (ADO Connection Object)

Using the **Open** method on a **Connection** object establishes the physical connection to a data source. After this method successfully completes, the connection is live and you can issue commands against it and process results.

Use the optional *ConnectionString* argument to specify a connection string containing a series of *argument* = *value* statements separated by semicolons. The **ConnectionString** property

automatically inherits the value used for the *ConnectionString* argument. Therefore, you can either set the **ConnectionString** property of the **Connection** object before opening it, or use the *ConnectionString* argument to set or override the current connection parameters during the **Open** method call.

If you pass user and password information both in the *ConnectionString* argument and in the optional *UserID* and *Password* arguments, the results may be unpredictable; you should only pass such information in either the *ConnectionString* argument or the *UserID* and *Password* arguments.

When you have concluded your operations over an open **Connection**, use the ADO Connection Object Close Method to free any associated system resources. Closing an object does not remove it from memory; you may change its property settings and use the **Open** method to open it again later. To completely eliminate an object from memory, set the object variable to *Nothing*.

Open Method Examples (ADO Connection Object)

See the ADO Connection Object Close Method.

ADO Connection Object BeginTrans, CommitTrans, and RollbackTrans Methods

The transaction methods manage transaction processing within a **Connection** object.

These transaction methods are summarized as follows:

Method	Description
BeginTrans	Begins a new transaction
CommitTrans	Saves any changes and ends the current transaction. It may also start a new transaction.
RollbackTrans	Cancels any changes made during the current transaction and ends the transaction. It may also start a new transaction.

BeginTrans, CommitTrans, and RollbackTrans Methods Syntax

```
level = connection.BeginTrans()

connection.BeginTrans

connection.CommitTrans

connection.RollbackTrans
```

BeginTrans, CommitTrans, and RollbackTrans Methods Remarks

Use these methods with a **Connection** object when you want to save or cancel a series of changes made to the source data as a single unit. For example, to transfer money between accounts, you subtract an amount from one and add the same amount to the other. If either update fails, the accounts no longer balance. Making these changes within an open transaction ensures either all or none of the changes goes through.

Not all providers support transactions. Check that the provider-defined property "Transaction DDL" appears in the **Connection** object's ADO Properties Collection, indicating that the provider

supports transactions. If the provider does not support transactions, calling one of these methods will return an error.

Once you call the **BeginTrans** method, the provider will no longer instantaneously commit any changes you make until you call **CommitTrans** or **RollbackTrans** to end the transaction.

For providers that support nested transactions, calling the **BeginTrans** method within an open transaction starts a new, nested transaction. The return value indicates the level of nesting: a return value of "1" indicates you have opened a top-level transaction (that is, the transaction is not nested within another transaction), "2" indicates that you have opened a second-level transaction (a transaction nested within a top-level transaction), and so forth. Calling **CommitTrans** or **RollbackTrans** affects only the most recently opened transaction; you must close or rollback the current transaction before you can resolve any higher-level transactions.

Calling the **CommitTrans** method saves changes made within an open transaction on the connection and ends the transaction. Calling the **RollbackTrans** method reverses any changes made within an open transaction and ends the transaction. Calling either method when there is no open transaction generates an error.

Depending on the **Connection** object's ADO Connection Object Attributes Property, calling either the **CommitTrans** or **RollbackTrans** methods may automatically start a new transaction. If the **Attributes** property is set to **adXactCommitRetaining**, the provider automatically starts a new transaction after a **CommitTrans** call. If the **Attributes** property is set to **adXactAbortRetaining**, the provider automatically starts a new transaction after a **RollbackTrans** call.

BeginTrans, CommitTrans, and RollbackTrans Methods Return Value

BeginTrans can be called as a function that returns a **Long** variable indicating the nesting level of the transaction.

BeginTrans, CommitTrans, and RollbackTrans Methods Examples

This Visual Basic example changes the book type of all psychology books in the ***Titles*** table of the database. After the **BeginTrans** method starts a transaction that isolates all the changes made to the ***Titles*** table, the **CommitTrans** method saves the changes. Notice that you can use the **RollbackTrans** method to undo changes that you saved using the ADO Recordset Object Update Method.

```
Public Sub BeginTransX()

Dim cnn1 As ADODB.Connection

Dim rstTitles As ADODB.Recordset

Dim strCnn As String

Dim strTitle As String

Dim strMessage As String

` Open connection.

strCnn = "driver={SQL Server};server=srv;" & _
```

```
"uid=sa;pwd=;database=pubs"
Set cnn1 = New ADODB.Connection
cnn1.Open strCnn
` Open titles table.
Set rstTitles = New ADODB.Recordset
rstTitles.CursorType = adOpenDynamic
rstTitles.LockType = adLockPessimistic
rstTitles.Open "titles", cnn1, , , adCmdTable
rstTitles.MoveFirst
cnn1.BeginTrans
` Loop through recordset and ask user if she wants
` to change the type for a specified title.
Do Until rstTitles.EOF
If Trim(rstTitles!Type) = "psychology" Then
strTitle = rstTitles!Title
strMessage = "Title: " & strTitle & vbCr & _
"Change type to self help?"
` Change the title for the specified employee.
If MsgBox(strMessage, vbYesNo) = vbYes Then
rstTitles!Type = "self_help"
rstTitles.Update
End If
End If
rstTitles.MoveNext
Loop
` Ask if the user wants to commit to all the
` changes made above.
If MsgBox("Save all changes?", vbYesNo) = vbYes Then
cnn1.CommitTrans
Else
cnn1.RollbackTrans
End If
```

```
` Print current data in recordset.
rstTitles.Requery
rstTitles.MoveFirst
Do While Not rstTitles.EOF
Debug.Print rstTitles!Title & " - " & rstTitles!Type
rstTitles.MoveNext
Loop
' Restore original data
rstTitles.MoveFirst
Do Until rstTitles.EOF
If Trim(rstTitles!Type) = "self_help" Then
rstTitles!Type = "psychology"
rstTitles.Update
End If
rstTitles.MoveNext
Loop
rstTitles.Close
cnn1.Close
End Sub
```

ADO Connection Object Properties

ADO Connection Object Attributes Property

One or more characteristics of an object. *This property is read-only on UNIX.*

Attributes Property Return Values (ADO Connection Object)

Sets or returns a **Long** value.

For a **Connection** object, the **Attributes** property is read/write, and its value can be the sum of any one or more of these **XactAttributeEnum** values (default is zero):

Value	Description
adXactCommitRetaining	Performs retaining commits, that is, calling the **CommitTrans** method automatically starts a new transaction. Not all providers support this, and it is always zero under UNIX.
adXactAbortRetaining	Performs retaining aborts, that is, calling the **BeginTrans, CommitTrans, and RollbackTrans** methods automatically

starts a new transaction. Not all providers support this, and it is always zero under UNIX.

Attributes Property Remarks (ADO Connection Object)

Use the **Attributes** property to set or return characteristics of **Connection** objects.

When you set multiple attributes, you can sum the appropriate constants. If you set the property value to a sum including incompatible constants, an error occurs.

Attributes Property Examples (ADO Connection Object)

This Visual Basic example displays the value of the **Attributes** property for **Connection** objects. It uses the ADO Field Object Name Property to display the name of each **Field** and **Property** object.

```
Public Sub AttributesX

Dim cnn1 As ADODB.Connection

Dim strCnn As String

' Open connection

strCnn = "driver={SQL Server};server=srv;" & _

"uid=sa;pwd=;database=pubs"

Set cnn1 = New ADODB.Connection

cnn1.Open strCnn

' Display the attributes of the connection.

Debug.Print "Connection attributes = " & _

cnn1.Attributes

cnn1.Close

End Sub
```

ADO Connection Object CommandTimeout Property

How long to wait while executing a command before terminating the attempt and generating an error.

CommandTimeout Property Return Values (ADO Connection Object)

Sets or returns a **Long** value that specifies, in seconds, how long to wait for a command to execute. Default is 30.

CommandTimeout Property Remarks (ADO Connection Object)

Use the **CommandTimeout** property on a **Connection** object to allow the cancellation of an ADO Connection Object Execute Method call, due to delays from network traffic or heavy server use. If the interval set in the **CommandTimeout** property elapses before the command completes

execution, an error occurs and ADO cancels the command. If you set the property to zero, ADO will wait indefinitely until the execution is complete. Make sure the provider and data source to which you are writing code supports the **CommandTimeout** functionality.

The **CommandTimeout** setting on a **Connection** object has no effect on the **CommandTimeout** setting on a **Command** object on the same **Connection**; that is, the **Command** object's **CommandTimeout** property does not inherit the value of the **Connection** object's **CommandTimeout** value.

On a **Connection** object, the **CommandTimeout** property remains read/write after the **Connection** is opened.

CommandTimeout Property Examples (ADO Connection Object)

See the **ActiveConnection** property.

ADO Connection Object ConnectionString Property

Contains the information used to establish a connection to a data source.

ConnectionString Property Return Values (ADO Connection Object)

Sets or returns a **String** value.

ConnectionString Property Remarks (ADO Connection Object)

Use the **ConnectionString** property to specify a data source by passing a detailed connection string containing a series of *argument* = *value* statements separated by semicolons.

ADO supports seven arguments for the **ConnectionString** property; any other arguments pass directly to the provider without any processing by ADO. The arguments ADO supports are as follows:

Argument	Description
Provider	Specifies the name of the provider to use for the connection.
DataSource	Specifies the name of a data source for the connection, for example, an Oracle database registered as an ODBC data source.
UserID	Specifies the user name to use when opening the connection.
Password	Specifies the password to use when opening the connection.
FileName	Specifies the name of a provider-specific file (for example, a persisted data source object) containing preset connection information.
RemoteProvider	Specifies the name of a provider to use when opening a client-side connection (Remote Data Service only).
RemoteServer	Specifies the path name of the server to use when opening a client-side connection (Remote Data Service only).

After you set the **ConnectionString** property and open the **Connection** object, the provider may alter the contents of the property, for example, by mapping the ADO-defined argument names to their provider equivalents.

The **ConnectionString** property automatically inherits the value used for the *ConnectionString* argument of the ADO Connection Object Open Method, so you can override the current **ConnectionString** property during the **Open** method call.

Because the *File Name* argument causes ADO to load the associated provider, you cannot pass both the *Provider* and *File Name* arguments.

The **ConnectionString** property is read/write when the connection is closed and read-only when it is open.

Remote Data Service Usage: When used on a client-side **Connection** object, the **ConnectionString** property can only include the *Remote Provider* and *Remote Server* parameters.

ConnectionString Property Example (ADO Connection Object)

This Visual Basic example demonstrates different ways of using the **ConnectionString** property to open a **Connection** object. It also uses the **ConnectionTimeout** property to set a connection timeout period, and the ADO Connection Object State Property to check the state of the connections. The **GetState** function is required for this procedure to run.

```
Public Sub ConnectionStringX()

Dim cnn1 As ADODB.Connection

Dim cnn2 As ADODB.Connection

Dim cnn3 As ADODB.Connection

Dim cnn4 As ADODB.Connection

' Open a connection without using a Data Source Name (DSN).

Set cnn1 = New ADODB.Connection

cnn1.ConnectionString = "driver={SQL Server};" & _

"server=bigsmile;uid=sa;pwd=pwd;database=pubs"

cnn1.ConnectionTimeout = 30

cnn1.Open

' Open a connection using a DSN and ODBC tags.

Set cnn2 = New ADODB.Connection

cnn2.ConnectionString = "DSN=Pubs;UID=sa;PWD=pwd;"

cnn2.Open

' Open a connection using a DSN and OLE DB tags.

Set cnn3 = New ADODB.Connection

cnn3.ConnectionString = "Data Source=Pubs;User ID=sa;Password=pwd;"
```

```
cnn3.Open
' Open a connection using a DSN and individual
' arguments instead of a connection string.
Set cnn4 = New ADODB.Connection
cnn4.Open "Pubs", "sa", "pwd"
' Display the state of the connections.
MsgBox "cnn1 state: " & GetState(cnn1.State) & vbCr & _
"cnn2 state: " & GetState(cnn1.State) & vbCr & _
"cnn3 state: " & GetState(cnn1.State) & vbCr & _
"cnn4 state: " & GetState(cnn1.State)
cnn4.Close
cnn3.Close
cnn2.Close
cnn1.Close
End Sub
Public Function GetState(intState As Integer) As String
Select Case intState
Case adStateClosed
GetState = "adStateClosed"
Case adStateOpen
GetState = "adStateOpen"
End Select
End Function
```

ADO Connection Object ConnectionTimeout Property

Sets how long to wait while establishing a connection before terminating the attempt and generating an error.

ConnectionTimeout Property Return Values (ADO Connection Object)

Sets or returns a **Long** value that specifies, in seconds, how long to wait for the connection to open. Default is 15.

ConnectionTimeout Property Remarks (ADO Connection Object)

Use the **ConnectionTimeout** property on a **Connection** object if delays from network traffic or heavy server use make it necessary to abandon a connection attempt. If the time from the

ConnectionTimeout property setting elapses prior to the opening of the connection, an error occurs and ADO cancels the attempt. If you set the property to zero, ADO will wait indefinitely until the connection is opened. Make sure the provider to which you are writing code supports the **ConnectionTimeout** functionality.

The **ConnectionTimeout** property is read/write when the connection is closed and read-only when it is open.

ConnectionTimeout Property Example (ADO Connection Object)

See the **ConnectionString** property.

ADO Connection Object CursorLocation Property

Sets or returns the location of the cursor engine.

CursorLocation Property Return Values (ADO Connection Object)

Sets or returns a **Long** value that can be set to one of the following constants:

Constant	Description
adUseClient	Uses client-side cursors supplied by a local cursor library. Local cursor engines will often allow many features that driver-supplied cursors may not, so using this setting may provide an advantage with respect to features that will be enabled. For backward-compatibility, the synonym adUseClientBatch is also supported. **Note:** With the Sun Chili!Soft ASP implementation of ADO, adUseClient has a value of 1, and adUseClientBatch has a value of 3.
adUseServer	Default. Uses data provider or driver-supplied cursors. These cursors are sometimes very flexible and allow for some additional sensitivity to reflecting changes that others make to the actual data source. However, some features of the Microsoft Client Cursor Provider (such as disassociated recordsets) cannot be simulated. **Note:** With the Sun Chili!Soft ASP implementation of ADO, adUseServer has a value of 2.

CursorLocation Property Remarks (ADO Connection Object)

This property allows you to choose between various cursor libraries accessible to the provider. Usually, you can choose between using a client-side cursor library or one that is located on the server.

This property setting only affects connections established after the property has been set. Changing the **CursorLocation** property has no effect on existing connections.

This property is read/write on a **Connection**.

CursorLocation Property Example (ADO Connection Object)

See the **AbsolutePosition** property example.

ADO Connection Object DefaultDatabase Property

The default database for a **Connection** object.

DefaultDatabase Property Return Values

Sets or returns a **String** that evaluates to the name of a database available from the provider.

DefaultDatabase Property Remarks

Use the **DefaultDatabase** property to set or return the name of the default database on a specific **Connection** object.

If there is a default database, SQL strings may use an unqualified syntax to access objects in that database. To access objects in a database other than the one specified in the **DefaultDatabase** property, you must qualify object names with the desired database name. Upon connection, the provider will write default database information to the **DefaultDatabase** property. Some providers allow only one database per connection, in which case you cannot change the **DefaultDatabase** property.

Some data sources and providers may not support this feature, and may return an error or an empty string.

Remote Data Service Usage: This property is not available on a client-side **Connection** object.

DefaultDatabase Property Example

See ADO Connection Object Provider Property

ADO Connection Object IsolationLevel Property

The level of transaction isolation for a **Connection** object. *Transactions are not currently supported on UNIX.*

IsolationLevel Property Return Values

Sets or returns one of the following **IsolationLevelEnum** values:

Constant	Description
adXactUnspecified	The provider is using a different **IsolationLevel** than specified, but the level cannot be determined.
adXactChaos	You cannot overwrite pending changes from more highly isolated transactions.
adXactBrowse	You can view uncommitted changes from one transaction in other transactions.
adXactReadUncommitted	Same as adXactBrowse.
adXactCursorStability	Default. You can view changes in other transactions

	only after they have been committed.
adXactReadCommitted	Same as adXactCursorStability.
adXactRepeatableRead	You cannot see changes in other transactions, but requerying can bring new **Recordset** objects.
adXactIsolated	Transactions are conducted in isolation of other transactions.
adXactSerializable	Same as adXactIsolated.

IsolationLevel Property Remarks

Use the **IsolationLevel** property to set the isolation level of a **Connection** object. The **IsolationLevel** property is read/write. The setting does not take effect until the next time you call the **BeginTrans** method. If the level of isolation you request is unavailable, the provider may return the next greater level of isolation.

IsolationLevel Property Example

This example uses the ADO Connection Object Mode Property to open an exclusive connection, and the **IsolationLevel** property to open a transaction that is conducted in isolation of other transactions.

```
Public Sub IsolationLevelX()
Dim cnn1 As ADODB.Connection
Dim rstTitles As ADODB.Recordset
Dim strCnn As String
` Assign connection string to variable.
strCnn = "driver={SQL Server};server=srv;" & _
"uid=sa;pwd=;database=pubs"
` Open connection and titles table.
Set cnn1 = New ADODB.Connection
cnn1.Mode = adModeShareExclusive
cnn1.IsolationLevel = adXactIsolated
cnn1.Open strCnn
Set rstTitles = New ADODB.Recordset
rstTitles.CursorType = adOpenDynamic
rstTitles.LockType = adLockPessimistic
rstTitles.Open "titles", cnn1, , , adCmdTable
cnn1.BeginTrans
` Display connection mode.
```

```
If cnn1.Mode = adModeShareExclusive Then
MsgBox "Connection mode is exclusive."
Else
MsgBox "Connection mode is not exclusive."
End If
` Display isolation level.
If cnn1.IsolationLevel = adXactIsolated Then
MsgBox "Transaction is isolated."
Else
MsgBox "Transaction is not isolated."
End If
` Change the type of psychology titles.
Do Until rstTitles.EOF
If Trim(rstTitles!Type) = "psychology" Then
rstTitles!Type = "self_help"
rstTitles.Update
End If
rstTitles.MoveNext
Loop
` Print current data in recordset.
rstTitles.Requery
Do While Not rstTitles.EOF
Debug.Print rstTitles!Title & " - " & rstTitles!Type
rstTitles.MoveNext
Loop
` Restore original data.
cnn1.RollbackTrans
rstTitles.Close
cnn1.Close
End Sub
```

ADO Connection Object Mode Property

The available permissions for modifying data in a **Connection**.

Mode Property Return Values (ADO Connection Object)

Sets or returns one of the following **ConnectModeEnum** values:

Constant	Description
adModeUnknown	Default. The permissions have not been set or cannot be determined.
adModeRead	Read-only permission.
adModeWrite	Write-only permission.
adModeReadWrite	Read/write permission.
adModeShareDenyRead	Prevents others from opening a connection with read permission.
adModeShareDenyWrite	Prevents others from opening a connection with write permission.
adModeShareExclusive	Prevents others from opening a connection.
adModeShareDenyNone	Allows others to open a connection with any permissions. Neither read nor write access can be denied to others.

Mode Property Remarks (ADO Connection Object)

Use the **Mode** property to set or return the access permissions in use by the provider on the current connection. You can set the **Mode** property only when the **Connection** object is closed.

Mode Property Example (ADO Connection Object)

See the **IsolationLevel** property example.

ADO Connection Object Provider Property

The name of the provider for a **Connection** object. *This property is not available on UNIX.*

Provider Property Return Values

Sets or returns a **String** value.

Provider Property Remarks

Use the **Provider** property to set or return the name of the provider for a connection. This property can also be set by the contents of the **ConnectionString** property or the *ConnectionString* argument of the ADO Connection Object Open Method; however, specifying a provider in more than one place while calling the **Open** method can have unpredictable results. If no provider is specified, the property will default to MSDASQL (Microsoft OLE DB Provider for ODBC).

The **Provider** property is read/write when the connection is closed and read-only when it is open. The setting does not take effect until you either open the **Connection** object or access the ADO Properties Collection of the **Connection** object. If the setting is invalid, an error occurs.

Provider Property Example

This Visual Basic example demonstrates the **Provider** property by opening two **Connection** objects using different providers. It also uses the **DefaultDatabase** property to set the default database for the Microsoft ODBC Provider.

```
Public Sub ProviderX()
Dim cnn1 As ADODB.Connection
Dim cnn2 As ADODB.Connection
` Open a connection using the Microsoft ODBC provider.
Set cnn1 = New ADODB.Connection
cnn1.ConnectionString = "driver={SQL Server};" & _
"server=bigsmile;uid=sa;pwd=pwd"
cnn1.Open strCnn
cnn1.DefaultDatabase = "pubs"
` Display the provider.
MsgBox "Cnn1 provider: " & cnn1.Provider
` Open a connection using the Microsoft Jet provider.
Set cnn2 = New ADODB.Connection
cnn2.Provider = "Microsoft.Jet.OLEDB.3.51"
cnn2.Open "C:\Samples\northwind.mdb", "admin", ""
` Display the provider.
MsgBox "Cnn2 provider: " & cnn2.Provider
cnn1.Close
cnn2.Close
End Sub
```

ADO Connection Object State Property

Describes the current state of an object.

State Property Return Values (ADO Connection Object)

Sets or returns a **Long** value that can be one of the following constants:

Constant	Description
adStateClosed	Default. The object is closed.
adStateOpen	The object is open.

State Property Remarks (ADO Connection Object)

You can use the **State** property to determine the current state of a given object at any time.

State Property Examples (ADO Connection Object)

This Visual Basic example demonstrates different ways of using the **ConnectionString** property to open a **Connection** object. It also uses the **ConnectionTimeout** property to set a connection timeout period, and the **State** property to check the state of the connections. The **GetState** function is required for this procedure to run.

```
Public Sub ConnectionStringX()
Dim cnn1 As ADODB.Connection
Dim cnn2 As ADODB.Connection
Dim cnn3 As ADODB.Connection
Dim cnn4 As ADODB.Connection
` Open a connection without using a DSN.
Set cnn1 = New ADODB.Connection
cnn1.ConnectionString = "driver={SQL Server};" & _
"server=bigsmile;uid=sa;pwd=pwd;database=pubs"
cnn1.ConnectionTimeout = 30
cnn1.Open
` Open a connection using a DSN and ODBC tags.
Set cnn2 = New ADODB.Connection
cnn2.ConnectionString = "DSN=Pubs;UID=sa;PWD=pwd;"
cnn2.Open
` Open a connection using a DSN and OLE DB tags.
Set cnn3 = New ADODB.Connection
cnn3.ConnectionString = "Data Source=Pubs;User ID=sa;Password=pwd;"
cnn3.Open
` Open a connection using a DSN and individual
` arguments instead of a connection string.
Set cnn4 = New ADODB.Connection
cnn4.Open "Pubs", "sa", "pwd"
```

```
` Display the state of the connections.
MsgBox "cnn1 state: " & GetState(cnn1.State) & vbCr &_
"cnn2 state: " & GetState(cnn1.State) & vbCr & _
"cnn3 state: " & GetState(cnn1.State) & vbCr & _
"cnn4 state: " & GetState(cnn1.State)
cnn4.Close
cnn3.Close
cnn2.Close
cnn1.Close
End Sub
Public Function GetState(intState As Integer) As String
Select Case intState
Case adStateClosed
GetState = "adStateClosed"
Case adStateOpen
GetState = "adStateOpen"
End Select
End Function
```

ADO Connection Object Version Property

The ADO version number.

Version Property Return Values

Returns a **String** value.

Version Property Remarks

Use the **Version** property to return the version number of the ADO implementation. The version of the provider will be available on Windows servers as a dynamic property in the ADO Properties Collection. *The **Properties** collection is not currently supported on UNIX.*

Version Property Example

This Visual Basic example uses the **Version** property of a **Connection** object to display the current ADO version. It also uses several dynamic properties to show the current DBMS name and version, OLE DB version, provider name and version, driver name and version, and driver ODBC version.

```
Public Sub VersionX()
```

```
Dim cnn1 As ADODB.Connection
' Open connection.
Set cnn1 = New ADODB.Connection
strCnn = "driver={SQL Server};server=srv;" & _
"uid=sa;pwd=;database=pubs"
cnn1.Open strCnn
strVersionInfo = "ADO Version: " & cnn1.Version & vbCr & _
"DBMS Name: " & cnn1.Properties("DBMS Name") & vbCr & _
"DBMS Version: " & cnn1.Properties("DBMS Version") & vbCr & _
"OLE DB Version: " & cnn1.Properties("OLE DB Version") & vbCr & _
"Provider Name: " & cnn1.Properties("Provider Name") & vbCr & _
"Provider Version: " & cnn1.Properties("Provider Version") & vbCr &
_
"Driver Name: " & cnn1.Properties("Driver Name") & vbCr & _
"Driver Version: " & cnn1.Properties("Driver Version") & vbCr & _
"Driver ODBC Version: " & cnn1.Properties("Driver ODBC Version")
MsgBox strVersionInfo
cnn1.Close
End Sub
```

ADO Error Object

ADO Error Object

The ADO **Error** object provides specific details about each ADO error.

ADO Error Object Properties

ADO Error Object Description Property	A descriptive string associated with an error.
ADO Error Object HelpContext, HelpFile Property	The help file topic associated with an error.
ADO Error Object HelpContext, HelpFile Property	The help file associated with an error.
ADO Error Object NativeError Property	The provider-specific error code for an error.
ADO Error Object Number Property	The number that uniquely identifies an

	error.
ADO Error Object Source Property	The name of the object or application that originally generated the error.
ADO Error Object SQLState Property	The SQL state for a given error.

ADO Error Object Remarks

Any operation involving ADO objects can generate one or more provider errors. As each error occurs, one or more **Error** objects are placed in the ADO Errors Collection of the ADO Connection Object. When another ADO operation generates an error, the **Errors** collection is cleared, and the new set of **Error** objects are placed in the **Errors** collection.

Note

> Each **Error** object represents a specific provider error, not an ADO error. ADO errors are exposed to the run-time exception handling mechanism. For example, in Microsoft Visual Basic, the occurrence of an ADO-specific error will trigger an **On Error** event and appear in the **Err** object. For a complete list of ADO errors, see Appendix B.

Read the **Error** object's properties to obtain specific details about each error:

- The ADO Error Object Description Property contains the text of the error.
- The ADO Error Object Number Property contains the **Long** integer value of the error constant.
- The ADO Error Object Source Property identifies the object that raised the error. This is particularly useful when you have several **Error** objects in the **Errors** collection following a request to a data source.
- The ADO Error Object HelpContext, and HelpFile Properties indicate the appropriate Microsoft Windows Help file and Help topic, respectively (if any exist), for the error.
- The ADO Error Object SQLState Property and ADO Error Object NativeError Property properties provide information from SQL data sources.

ADO supports the return of multiple errors by a single ADO operation to allow for error information specific to the provider. To obtain this error information in an error handler, use the appropriate error-trapping features of the language or environment you are working with, then use nested loops to enumerate the properties of each **Error** object in the **Errors** collection.

ADO clears the **OLE Error Info** object before making a call that could potentially generate a new provider error. However, the **Errors** collection on the **Connection** object is cleared and populated only when the provider generates a new error, or when the ADO Collections Clear Method is called.

Some properties and methods return warnings that appear as **Error** objects in the **Errors** collection but do not halt a program's execution. Before you call the ADO Recordset Object Resync Method**,** ADO Recordset Object UpdateBatch Method, or ADO Recordset Object CancelBatch Method methods on an ADO Recordset Object, or before you set the ADO Recordset Object Filter Property on a **Recordset** object, call the ADO Collections Clear Method

on the **Errors** collection so that you can read the Count property of the **Errors** collection to test for returned warnings.

If there is no valid **Connection** object when using Microsoft Visual Basic and VBScript, retrieve error information from the **Err** object.

To refer to an **Error** object in a collection by its ordinal number, use either of the following syntax forms:

```
connection.Errors.Item(0)

connection.Errors(0)
```

ADO Error Object Properties

ADO Error Object Description Property

A descriptive string associated with an **Error** object.

Description Property Return Values (ADO Error Object)

Returns a **String** value.

Description Property Remarks (ADO Error Object)

Use the **Description** property to obtain a short description of the error. Display this property to alert the user to an error that you cannot or do not want to handle. The string will come from either ADO or a provider.

Providers are responsible for passing specific error text to ADO. ADO adds an **Error** object to the ADO Errors Collection for each provider error or warning it receives. Enumerate the **Errors** collection to trace the errors that the provider passes.

Description Property Example (ADO Error Object)

This Visual Basic example triggers an error, traps it, and displays the ADO Error Object Description Property, ADO Error Object HelpContext, and HelpFile Properties, ADO Error Object NativeError Property**,** ADO Error Object Number Property, ADO Error Object Source Property, and ADO Error Object SQLState Property properties of the resulting **Error** object:

```
Public Sub DescriptionX()

Dim cnn1 As ADODB.Connection

Dim errLoop As ADODB.Error

Dim strError As String

On Error GoTo ErrorHandler

` Intentionally trigger an error.

Set cnn1 = New ADODB.Connection

cnn1.Open "nothing"

Exit Sub
```

```
ErrorHandler:
` Enumerate Errors collection and display
` properties of each Error object.
For Each errLoop In cnn1.Errors
strError = "Error #" & errLoop.Number & vbCr & _
" " & errLoop.Description & vbCr & _
" (Source: " & errLoop.Source & ")" & vbCr & _
" (SQL State: " & errLoop.SQLState & ")" & vbCr & _
" (NativeError: " & errLoop.NativeError & ")" & vbCr
If errLoop.HelpFile = "" Then
strError = strError & _
" No Help file available" & _
vbCr & vbCr
Else
strError = strError & _
" (HelpFile: " & errLoop.HelpFile & ")" & vbCr & _
" (HelpContext: " & errLoop.HelpContext & ")" & _
vbCr & vbCr
End If
Debug.Print strError
Next
Resume Next
End Sub
```

ADO Error Object HelpContext, HelpFile Property

The help file and topic associated with an **Error** object.

HelpContext, HelpFile Property Return Values

HelpContextID

Returns a context ID, as a **Long** value, for a topic in a Microsoft Windows Help file.

HelpFile

Returns a **String** that evaluates to a fully resolved path to a Help file.

HelpContext, HelpFile Property Remarks

If a Windows Help (.hlp) file is specified in the **HelpFile** property, the **HelpContext** property is used to automatically display the Help topic it identifies. If there is no relevant help topic available, the **HelpContext** property returns zero and the **HelpFile** property returns a zero-length string ("").

HelpContext, HelpFile Property Examples

See the ADO Error Object Description Property example.

ADO Error Object NativeError Property

The provider-specific error code for a given **Error** object.

NativeError Property Return Values

Returns a **Long** value.

NativeError Property Remarks

Use the **NativeError** property to retrieve the database-specific error information for a particular **Error** object. For example, when using the Microsoft ODBC Provider for OLE DB with a SQL Server database, native error codes that originate from SQL Server pass through ODBC and the ODBC Provider to the ADO **NativeError** property.

NativeError Property Example

See the ADO Error Object Description Property.

ADO Error Object Number Property

The number that uniquely identifies an **Error** object.

Number Property Return Values

Returns a **Long** value.

Number Property Remarks

Use the **Number** property to determine which error occurred. The value of the property is a unique number that corresponds to the error condition.

Number Property Example

See the ADO Error Object Description Property.

ADO Error Object Source Property

The name of the object or application that originally generated an error.

Source Property Return Values

Returns a **String** value.

Source Property Remarks

Use the **Source** property on an **Error** object to determine the name of the object or application that originally generated an error. This could be the object's class name or programmatic ID. For errors in ADODB, the property value will be **ADODB.***ObjectName.*

Source Property Parameters (ADO Error Object)

ObjectName

The name of the object that triggered the error. The **Source** property is read-only for **Error** objects.

Based on the error documentation from the **Source**, ADO Error Object Number Property, and ADO Error Object Description Property properties of **Error** objects, you can write code that will handle the error appropriately.

Source Property Example

See the ADO Error Object Description Property example.

ADO Error Object SQLState Property

The SQL state for a given **Error** object.

SQLState Property Return Values

Returns a five-character **String** that follows the ANSI SQL standard.

SQLState Property Remarks

Use the **SQLState** property to read the five-character error code that the provider returns when an error occurs during the processing of a SQL statement. For example, when using the Microsoft OLE DB Provider for ODBC with a SQL Server database, SQL state error codes originate from ODBC based either on errors specific to ODBC or on errors that originate from Microsoft SQL Server, and are then mapped to ODBC errors. These error codes are documented in the ANSI SQL standard, but may be implemented differently by different data sources.

SQLState Property Example

See the ADO Error Object Description Property example.

ADO Field Object

ADO Field Object

The ADO Field Object represents a column of data with a common data type.

ADO Field Object Collections

ADO Properties Collection	All **Property** objects for a specific instance of a **Field** object. *This collection is not currently supported on UNIX.*

ADO Field Object Methods

ADO Field Object AppendChunk Method	Appends data to a large text or binary data field.
ADO Field Object GetChunk Method	Returns all or a portion of the contents of a large text or binary data field.

ADO Field Object Properties

ADO Field Object ActualSize Property	The actual length of a field value.
ADO Field Object Attributes Property	One or more characteristics of a field. *This property is read-only on UNIX.*
ADO Field Object DefinedSize Property	The defined size of a field.
ADO Field Object Name Property	The name of a field.
ADO Field Object NumericScale Property	The scale of numeric values in a field.
ADO Field Object OriginalValue Property	The value of a field that existed in the record before any changes were made.
ADO Field Object Precision Property	The degree of precision for numeric values in a field.
ADO Field Object Type Property	The data type of the field.
ADO Field Object UnderlyingValue Property	The current value of the field in the database.
ADO Field Object Value Property	The value assigned to the field.

ADO Field Object Remarks

A ADO Recordset Object has an ADO Fields Collection made up of **Field** objects. Each **Field** object corresponds to a column in the recordset. You use the ADO Field Object Value Property of **Field** objects to set or return data for the current record. Depending on the functionality the provider exposes, some collections, methods, or properties of a **Field** object may not be available.

The collections, methods, and properties of a **Field** object are used to:

- return the name of a field with the ADO Field Object Name Property.
- view or change the data in the field with the ADO Field Object Value Property.
- return the basic characteristics of a field with the ADO Field Object Type Property, ADO Field Object Precision Property, and ADO Field Object NumericScale Property properties.
- return the declared size of a field with the ADO Field Object DefinedSize Property.
- return the actual size of the data in a given field with the ADO Field Object ActualSize Property.

- determine what types of functionality are supported for a given field with the ADO Field Object Attributes Property and ADO Properties Collection.
- manipulate the values of fields containing long binary or long character data with the ADO Field Object AppendChunk Method and ADO Field Object GetChunk Method methods.
- resolve discrepancies in field values during batch updating with the ADO Field Object OriginalValue Property and **UnderlyingValue** properties (if the provider supports batch updates).

Note

All metadata properties (**Name**, **Type**, **DefinedSize**, **Precision**, and **NumericScale**) are available before opening the **Field** object's recordset. Setting them at that time is useful for dynamically constructing forms.

ADO Field Object Methods

ADO Field Object AppendChunk Method

Appends data to a large text or binary data **Field** object.

AppendChunk Method Syntax (ADO Field Object)

```
object.AppendChunk Data
```

AppendChunk Method Parameters (ADO Field Object)

object

A **Field** object.

Data

A **Variant** containing the data you want to append to the object.

AppendChunk Method Remarks (ADO Field Object)

Use the **AppendChunk** method on a **Field** object to fill it with long binary or character data. In situations where system memory is limited, you can use the **AppendChunk** method to manipulate long values in portions rather than in their entirety.

If the **adFldLong** bit in the ADO Field Object Attributes Property of a **Field** object is set to **True**, you can use the **AppendChunk** method for that field.

The first **AppendChunk** call on a **Field** object writes data to the field, overwriting any existing data. Subsequent **AppendChunk** calls add to existing data. If you are appending data to one field and then you set or read the value of another field in the current record, ADO assumes that you are done appending data to the first field. If you call the **AppendChunk** method on the first field again, ADO interprets the call as a new **AppendChunk** operation and overwrites the existing data. Accessing fields in other ADO Recordset Object objects (that are not clones of the first **Recordset** object) will not disrupt **AppendChunk** operations.

If there is no current record when you call **AppendChunk** on a **Field** object, an error occurs.

AppendChunk Method Examples (ADO Field Object)

See the ADO Field Object GetChunk Method example.

ADO Field Object GetChunk Method

Returns all or a portion of the contents of a large text or binary data **Field** object.

GetChunk Method Syntax (ADO Field Object)

```
variable = field.GetChunk( Size )
```

GetChunk Method Parameters (ADO Field Object)

variable

Variant to hold data returned.

Size

A **Long** expression equal to the number of bytes or characters you want to retrieve.

GetChunk Method Remarks (ADO Field Object)

Use the **GetChunk** method on a **Field** object to retrieve part or all of its long binary or character data. In situations where system memory is limited, you can use the **GetChunk** method to manipulate long values in portions rather than in their entirety.

The data a **GetChunk** call returns is assigned to *variable*. If *Size* is greater than the remaining data, the **GetChunk** method returns only the remaining data without padding *variable* with empty spaces. If the field is empty, the **GetChunk** method returns **Null**.

Each subsequent **GetChunk** call retrieves data starting from where the previous **GetChunk** call left off. However, if you are retrieving data from one field and then you set or read the value of another field in the current record, ADO assumes you are done retrieving data from the first field. If you call the **GetChunk** method on the first field again, ADO interprets the call as a new **GetChunk** operation and starts reading from the beginning of the data. Accessing fields in other ADO Recordset Object objects (that are not clones of the first **Recordset** object) will not disrupt **GetChunk** operations.

If the **adFldLong** bit in the ADO Field Object Attributes Property of a **Field** object is set to **True**, you can use the **GetChunk** method for that field.

If there is no current record when you use the **GetChunk** method on a **Field** object, error 3021 (no current record) occurs.

GetChunk Method Return Values (ADO Field Object)

Returns a **Variant**.

GetChunk Method Example (ADO Field Object)

See the ADO Field Object AppendChunk Method.

ADO Field Object Properties

ADO Field Object ActualSize Property

The actual length of a field's value.

ActualSize Property Return Values (ADO Field Object)

Returns a **Long** value. Some providers may allow this property to be set to reserve space for BLOB data, in which case the default value is 0.

ActualSize Property Remarks (ADO Field Object)

Use the **ActualSize** property to return the actual length of a **Field** object's value. For all fields, the **ActualSize** property is read-only. If ADO cannot determine the length of the **Field** object's value, the **ActualSize** property returns **adUnknown**.

The **ActualSize** and ADO Field Object DefinedSize Property properties are different as shown in the following example: a **Field** object with a declared type of **adVarChar** and a maximum length of 50 characters returns a **DefinedSize** property value of 50, but the **ActualSize** property value it returns is the length of the data stored in the field for the current record.

ActualSize Property Example (ADO Field Object)

This Visual Basic example uses the **ActualSize** and **DefinedSize** properties to display the defined size and actual size of a field.

```
Public Sub ActualSizeX()

Dim rstStores As ADODB.Recordset

Dim strCnn As String

' Open a recordset for the Stores table.

strCnn = "driver={SQL Server};server=srv;" & _

"uid=sa;pwd=;database=pubs"

Set rstStores = New ADODB.Recordset

rstStores.Open "stores", strCnn, , , adCmdTable

' Loop through the recordset displaying the contents

' of the stor_name field, the field's defined size,

' and its actual size.

rstStores.MoveFirst

Do Until rstStores.EOF

MsgBox "Store name: " & rstStores!stor_name & _

vbCr & "Defined size: " & _

rstStores!stor_name.DefinedSize & _
```

```
vbCr & "Actual size: " & _
rstStores!stor_name.ActualSize & vbCr
rstStores.MoveNext
Loop
rstStores.Close
End Sub
```

ADO Field Object Attributes Property

One or more characteristics of an object. *This property is read-only on UNIX.*

Attributes Property Return Values (ADO Field Object)

Sets or returns a **Long** value.

Attributes Property Field (ADO Field Object)

For a **Field** object, the **Attributes** property is read-only, and its value can be the sum of any one or more of these **FieldAttributeEnum** values:

Value	Description
adFldMayDefer	The field is deferred; that is, the field values are not retrieved from the data source with the whole record, but only when you explicitly access them.
adFldUpdatable	The field can be written.
adFldUnknownUpdatable	The provider cannot determine if the field can be written.
adFldFixed	The field contains fixed-length data.
adFldIsNullable	The field accepts **Null** values.
adFldMayBeNull	You can read **Null** values from the field.
adFldLong	The field is a long binary field. Also indicates that you can use the ADO Field Object AppendChunk Method and ADO Field Object GetChunk Method methods.
adFldRowID	The field contains some kind of record ID (record number, unique identifier, and so forth).
adFldRowVersion	The field contains some kind of time or date stamp used to track updates.
adFldCacheDeferred	The provider caches field values and subsequent reads are done from the cache.

Attributes Property Remarks (ADO Field Object)

Use the **Attributes** property to set or return characteristics of **Field** objects.

When you set multiple attributes, you can sum the appropriate constants. If you set the property value to a sum including incompatible constants, an error occurs.

ADO Field Object DefinedSize Property

The defined size of a **Field** object.

DefinedSize Property Return Values (ADO Field Object)

Returns a **Long** value that reflects the defined size of a field as a number of bytes.

DefinedSize Property Remarks (ADO Field Object)

Use the **DefinedSize** property to determine the data capacity of a **Field** object.

The **DefinedSize** and ADO Field Object ActualSize Property properties are different. For example, consider a **Field** object with a declared type of **adVarChar** and a **DefinedSize** property value of 50, containing a single character. The **ActualSize** property value it returns is the length in bytes of the single character.

DefinedSize Property Examples (ADO Field Object)

See the ADO Field Object ActualSize Property example.

ADO Field Object Name Property

The name of an object.

Name Property Return Values (ADO Field Object)

Sets or returns a **String** value. The value is read-only on a **Field** object.

Name Property Remarks (ADO Field Object)

Use the **Name** property to retrieve the name of a **Field** object.

The **Name** property is read-only. Names do not have to be unique within a collection.

Name Property Examples (ADO Field Object)

See the ADO Field Object Attributes Property example.

ADO Field Object NumericScale Property

The scale of **Numeric** values in a **Field** object.

NumericScale Property Return Values (ADO Field Object)

Sets or returns a **Byte** value, indicating the number of decimal places to which numeric values will be resolved

NumericScale Property Remarks (ADO Field Object)

Use the **NumericScale** property to determine how many digits to the right of the decimal point will be used to represent values for a numeric **Field** object.

The **NumericScale** property is read-only.

ADO Field Object OriginalValue Property

The value of a **Field** object that existed in the record before any changes were made.

OriginalValue Property Return Values (ADO Field Object)

Returns a **Variant** value.

OriginalValue Property Remarks (ADO Field Object)

Use the **OriginalValue** property to return the original field value for a field from the current record.

In immediate update mode (the provider writes changes to the underlying data source once you call the ADO Recordset Object Update Method), the **OriginalValue** property returns the field value that existed prior to any changes (that is, since the last **Update** method call). This is the same value that the ADO Recordset Object CancelUpdate Method uses to replace the ADO Field Object Value Property.

In batch update mode (the provider caches multiple changes and writes them to the underlying data source only when you call the ADO Recordset Object UpdateBatch Method), the **OriginalValue** property returns the field value that existed prior to any changes (that is, since the last **UpdateBatch** method call). This is the same value that the ADO Recordset Object CancelBatch Method uses to replace the **Value** property. When you use this property with the **UnderlyingValue** property, you can resolve conflicts that arise from batch updates. Batch updates are currently not supported on UNIX.

OriginalValue Property Example (ADO Field Object)

This Visual Basic example demonstrates the **OriginalValue** and **UnderlyingValue** properties by displaying a message if a record's underlying data has changed during a ADO Recordset Object batch update.

```
Public Sub OriginalValueX()

Dim cnn1 As ADODB.Connection

Dim rstTitles As ADODB.Recordset

Dim fldType As ADODB.Field

Dim strCnn As String

' Open connection.

Set cnn1 = New ADODB.Connection

strCnn = "driver={SQL Server};server=srv;" & _

"uid=sa;pwd=;database=pubs"
```

```
cnn1.Open strCnn
' Open recordset for batch update.
Set rstTitles = New ADODB.Recordset
Set rstTitles.ActiveConnection = cnn1
rstTitles.CursorType = adOpenKeyset
rstTitles.LockType = adLockBatchOptimistic
rstTitles.Open "titles"
' Set field object variable for Type field.
Set fldType = rstTitles!Type
' Change the type of psychology titles.
Do Until rstTitles.EOF
If Trim(fldType) = "psychology" Then
fldType = "self_help"
End If
rstTitles.MoveNext
Loop
' Similate a change by another user by updating
' data using a command string.
cnn1.Execute "UPDATE titles SET type = 'sociology' " & _
"WHERE type = 'psychology'"
'Check for changes.
rstTitles.MoveFirst
Do Until rstTitles.EOF
If fldType.OriginalValue <> _
fldType.UnderlyingValue Then
MsgBox "Data has changed!" & vbCr & vbCr & _
" Title ID: " & rstTitles!title_id & vbCr & _
" Current value: " & fldType & vbCr & _
" Original value: " & _
fldType.OriginalValue & vbCr & _
" Underlying value: " & _
fldType.UnderlyingValue & vbCr
```

```
End If
rstTitles.MoveNext
Loop
' Cancel the update because this is a demonstration.
rstTitles.CancelBatch
rstTitles.Close
' Restore original values.
cnn1.Execute "UPDATE titles SET type = 'psychology' " & _
"WHERE type = 'sociology'"
cnn1.Close
End Sub
```

ADO Field Object Precision Property

The degree of precision for numeric **Field** objects.

Precision Property Return Values (ADO Field Object)

Sets or returns a **Byte** value, indicating the maximum total number of digits used to represent values. The value is read-only on a **Field** object.

Precision Property Remarks (ADO Field Object)

Use the **Precision** property to determine the maximum number of digits used to represent values for a numeric **Field** object.

Precision Property Example (ADO Field Object)

See the ADO Field Object NumericScale Property.

ADO Field Object Type Property

The operational type or data type of a **Field** object.

Type Property Return Values (ADO Field Object)

Sets or returns one of the following **DataTypeEnum** values. The corresponding OLE DB type indicators are as follows:

Constant	Description
adArray	Or'd together with another type to indicate that the data is a safe-array of that type (DBTYPE_ARRAY).
adBigInt	An 8-byte signed integer (DBTYPE_I8).
adBinary	A binary value (DBTYPE_BYTES).

adBoolean	A Boolean value (DBTYPE_BOOL).
adByRef	Or'd together with another type to indicate that the data is a pointer to data of the other type (DBTYPE_BYREF).
adBSTR	A null-terminated character string (Unicode) (DBTYPE_BSTR).
adChar	A String value (DBTYPE_STR).
adCurrency	A currency value (DBTYPE_CY). Currency is a fixed-point number with 4 digits to the right of the decimal point. It is stored in an 8-byte signed integer scaled by 10,000.
adDate	A date value (DBTYPE_DATE). A date is stored as a Double, the whole part of which is the number of days since December 30, 1899, and the fractional part of which is the fraction of a day.
adDBDate	A date value (yyyymmdd) (DBTYPE_DBDATE).
adDBTime	A time value (hhmmss) (DBTYPE_DBTIME).
adDBTimeStamp	A date-time stamp (yyyymmddhhmmss plus a fraction in billionths) (DBTYPE_DBTIMESTAMP).
adDecimal	An exact numeric value with a fixed precision and scale (DBTYPE_DECIMAL).
adDouble	A double-precision floating point value (DBTYPE_R8).
adEmpty	No value was specified (DBTYPE_EMPTY).
adError	A 32-bit error code (DBTYPE_ERROR).
adGUID	A globally unique identifier (GUID) (DBTYPE_GUID).
adIDispatch	A pointer to an **IDispatch** interface on an OLE object (DBTYPE_IDISPATCH).
adInteger	A 4-byte signed integer (DBTYPE_I4).
adIUnknown	A pointer to an **IUnknown** interface on an OLE object (DBTYPE_IUNKNOWN).
adNumeric	An exact numeric value with a fixed precision and scale (DBTYPE_NUMERIC).
adSingle	A single-precision floating point value (DBTYPE_R4).
adSmallInt	A 2-byte signed integer (DBTYPE_I2).
adTinyInt	A 1-byte signed integer (DBTYPE_I1).
adUnsignedBigInt	An 8-byte unsigned integer (DBTYPE_UI8).
adUnsignedInt	A 4-byte unsigned integer (DBTYPE_UI4).
adUnsignedSmallInt	A 2-byte unsigned integer (DBTYPE_UI2).
adUnsignedTinyInt	A 1-byte unsigned integer (DBTYPE_UI1).

adUserDefined	A user-defined variable (DBTYPE_UDT).
adVariant	An Automation **Variant** (DBTYPE_VARIANT).
adVector	OR'd together with another type to indicate that the data is a DBVECTOR structure, as defined by OLE DB, that contains a count of elements and a pointer to data of the other type (DBTYPE_VECTOR).
adWChar	A null-terminated Unicode character string (DBTYPE_WSTR).

Type Property Remarks (ADO Field Object)

For **Field** objects, the **Type** property is read-only.

Type Property Example (ADO Field Object)

This example demonstrates the **Type** property by displaying the name of the constant corresponding to the value of the **Type** property of all the **Field** objects in the ***Employees*** table. The **FieldType** function is required for this procedure to run.

```
Public Sub TypeX()
Dim rstEmployees As ADODB.Recordset
Dim fldLoop As ADODB.Field
Dim strCnn As String
` Open recordset with data from Employee table.
strCnn = "driver={SQL Server};server=srv;" & _
"uid=sa;pwd=;database=pubs"
Set rstEmployees = New ADODB.Recordset
rstEmployees.Open "employee", strCnn, , , adCmdTable
Debug.Print "Fields in Employee Table:" & vbCr
` Enumerate Fields collection of Employees table.
For Each fldLoop In rstEmployees.Fields
Debug.Print " Name: " & fldLoop.Name & vbCr & _
" Type: " & FieldType(fldLoop.Type) & vbCr
Next fldLoop
End Sub
Public Function FieldType(intType As Integer) As String
Select Case intType
Case adChar
FieldType = "adChar"
```

```
Case adVarChar
FieldType = "adVarChar"
Case adSmallInt
FieldType = "adSmallInt"
Case adUnsignedTinyInt
FieldType = "adUnsignedTinyInt"
Case adDBTimeStamp
FieldType = "adDBTimeStamp"
End Select
End Function
```

ADO Field Object UnderlyingValue Property

A **Field** object's current value in the database.

UnderlyingValue Property Return Values (ADO Field Object)

Returns a **Variant** value.

UnderlyingValue Property Remarks (ADO Field Object)

Use the **UnderlyingValue** property to return the current field value from the database. The field value in the **UnderlyingValue** property is the value that is visible to your transaction and may be the result of a recent update by another transaction. This may differ from the ADO Field Object OriginalValue Property, which reflects the value that was originally returned to the ADO Recordset Object.

This is similar to using the ADO Recordset Object Resync Method, but the **UnderlyingValue** property returns only the value for a specific field from the current record. This is the same value that the **Resync** method uses to replace the ADO Field Object Value Property.

When you use this property with the **OriginalValue** property, you can resolve conflicts that arise from batch updates.

UnderlyingValue Property Example (ADO Field Object)

See the ADO Field Object OriginalValue Property example.

ADO Field Object Value Property

Indicates the value assigned to a **Field** object.

Value Property Return Values (ADO Field Object)

Sets or returns a **Variant** value. Default value depends on the ADO Field Object Type Property.

Value Property Remarks (ADO Field Object)

Use the **Value** property to set or return data from **Field** objects. ADO allows setting and returning long binary data with the **Value** property.

Value Property Example (ADO Field Object)

This Visual Basic example demonstrates the **Value** property with **Field** and **Property** objects by displaying field and property values for the ***Employees*** table.

```
Public Sub ValueX()
Dim rstEmployees As ADODB.Recordset
Dim fldLoop As ADODB.Field
Dim prpLoop As ADODB.Property
Dim strCnn As String
' Open recordset with data from Employee table.
strCnn = "driver={SQL Server};server=srv;" & _
"uid=sa;pwd=;database=pubs"
Set rstEmployees = New ADODB.Recordset
rstEmployees.Open "employee", strCnn, , , adCmdTable
Debug.Print "Field values in rstEmployees"
' Enumerate the Fields collection of the Employees
' table.
For Each fldLoop In rstEmployees.Fields
` Because Value is the default property of a
` Field object, the use of the actual keyword
` here is optional.
Debug.Print " " & fldLoop.Name & " = " &
fldLoop.Value
Next fldLoop
Debug.Print "Property values in rstEmployees"
' Enumerate the Properties collection of the
' Recordset object.
For Each prpLoop In rstEmployees.Properties
' Because Value is the default property of a
' Property object, the use of the actual keyword
' here is optional.
Debug.Print " " & prpLoop.Name & " = " &
```

```
prpLoop.Value
Next prpLoop
rstEmployees.Close
End Sub
```

ADO Parameter Object

ADO Parameter Object

The ADO Parameter Object represents a parameter or argument associated with a **Command** object based on a parameterized query or stored procedure.

ADO Parameter Object Collections

ADO Properties Collection	All the **Property** objects for a specific instance of a **Parameter** object. *This collection is not currently supported on UNIX.*

ADO Parameter Object Methods

ADO Parameter Object AppendChunk Method	Appends data to a large text or binary data parameter.

ADO Parameter Object Properties

ADO Parameter Object Attributes Property	One or more characteristics of a parameter. *This property is currently read-only on UNIX.*
ADO Parameter Object Direction Property	Indicates if the parameter is an input parameter, an output parameter, or both; or if the parameter is the output of a stored procedure.
ADO Parameter Object Name Property	The name of the parameter.
ADO Parameter Object NumericScale Property	The scale of numeric values in the parameter.
ADO Parameter Object Precision Property	The degree of precision for numeric values in the parameter.
ADO Parameter Object Size Property	The maximum size, in bytes or characters, of a parameter.
ADO Parameter Object Type Property	The data type of the parameter.
ADO Parameter Object Value Property	The value assigned to the parameter.

ADO Parameter Object Remarks

*The **Properties** collection is not currently supported on UNIX.*

Many providers support *parameterized* commands. These are commands where the desired action is defined once, but variables (or parameters) are used to alter some details of the command. For

example, an SQL SELECT statement could use a parameter to define the matching criteria of a WHERE clause, and another to define the column name for a SORT BY clause.

The **Parameter** objects represent parameters associated with parameterized queries, or the in/out arguments and the return values of stored procedures. Depending on the functionality of the provider, some collections, methods, or properties of a **Parameter** object may not be available.

The collections, methods, and properties of a **Parameter** object are used to:

- set or return the name of a parameter with the ADO Parameter Object Name Property.
- set or return the value of a parameter with the ADO Parameter Object Value Property.
- set or return parameter characteristics with the ADO Parameter Object Attributes Property**,** ADO Parameter Object Direction Property**,** ADO Parameter Object Precision Property**,** ADO Parameter Object NumericScale Property, ADO Parameter Object Size Property**,** and ADO Parameter Object Type Property properties.
- pass long binary or character data to a parameter with the ADO Parameter Object AppendChunk Method.

If you know the names and properties of the parameters associated with the stored procedure or parameterized query you wish to call, you can use the **CreateParameter** method to create **Parameter** objects with the appropriate property settings and use the ADO Collections Append Method to add them to the ADO Parameters Collection. This lets you set and return parameter values without having to call the ADO Collections Refresh Method on the **Parameters** collection to retrieve the parameter information from the provider, a potentially resource-intensive operation.

ADO Parameter Object Methods

ADO Parameter Object AppendChunk Method

Appends data to a large text or binary data **Parameter** object.

AppendChunk Method Syntax (ADO Parameter Object)

```
object.AppendChunk Data
```

AppendChunk Method Parameters (ADO Parameter Object)

object

A **Parameter** object.

Data

A **Variant** containing the data you want to append to the object.

AppendChunk Method Remarks (ADO Parameter Object)

Use the **AppendChunk** method on a **Parameter** object to fill it with long binary or character data. In situations where system memory is limited, you can use the **AppendChunk** method to manipulate long values in portions rather than in their entirety.

If the **adFldLong** bit in the ADO Parameter Object Attributes Property of a **Parameter** object is set to **True**, you can use the **AppendChunk** method for that parameter.

The first **AppendChunk** call on a **Parameter** object writes data to the parameter, overwriting any existing data. Subsequent **AppendChunk** calls on a **Parameter** object adds to existing parameter data. An **AppendChunk** call that passes a **Null** value generates an error; you must manually set the ADO Parameter Object Value Property of the **Parameter** object to a zero-length string ("") in order to clear its value.

ADO Parameter Object Properties

ADO Parameter Object Attributes Property

One or more characteristics of an object. *This property is read-only on UNIX.*

Attributes Property Return Values (ADO Parameter Object)

Sets or returns a **Long** value.

Attributes Property Parameters (ADO Parameter Object)

For an ADO Parameter Object, the **Attributes** property is read/write, and its value can be the sum of any one or more of these **ParameterAttributesEnum** values:

Value	Description
adParamSigned	Default. The parameter accepts signed values.
adParamNullable	The parameter accepts **Null** values.
adParamLong	The parameter accepts long binary data.

Attributes Property Remarks (ADO Parameter Object)

Use the **Attributes** property to set or return characteristics of **Parameter** objects.

When you set multiple attributes, you can sum the appropriate constants. If you set the property value to a sum including incompatible constants, an error occurs.

Attributes Property Examples (ADO Parameter Object)

This Visual Basic example displays the value of the **Attributes** property for **Connection**, **Field**, and **Property** objects. It uses the ADO Parameter Object Name Property to display the name of each **Field** and **Property** object.

```
Public Sub AttributesX

Dim cnn1 As ADODB.Connection

Dim rstEmployees As ADODB.Recordset

Dim fldLoop As ADODB.Field

Dim proLoop As ADODB.Property

Dim strCnn As String
```

```
' Open connection and recordset.
strCnn = "driver={SQL Server};server=srv;" & _
"uid=sa;pwd=;database=pubs"
Set cnn1 = New ADODB.Connection
cnn1.Open strCnn
Set rstEmployees = New ADODB.Recordset
rstEmployees.Open "employee", cnn1, , ,
adCmdTable
' Display the attributes of the connection.
Debug.Print "Connection attributes = " & _
cnn1.Attributes
' Display attributes of the Employee table fields
Debug.Print "Field attributes:"
For Each fldLoop In rstEmployees.Fields
Debug.Print " " & fldLoop.Name & " = " & _
fldLoop.Attributes
Next fldLoop
' Display attributes of the Employee table properties.
Debug.Print "Property attributes:"
For Each proLoop In rstEmployees.Properties
Debug.Print " " & proLoop.Name & " = " & _
proLoop.Attributes
Next proLoop
rstEmployees.Close
cnn1.Close
End Sub
```

ADO Parameter Object Direction Property

Indicates whether the **Parameter** object represents an input parameter, an output parameter, or both, or if the parameter is the return value from a stored procedure.

Direction Property Return Values

Sets or returns one of the following **ParameterDirectionEnum** values

Constant	Description
AdParamInput	Default. Indicates an input parameter.
AdParamOutput	Indicates an output parameter.
AdParamInputOutput	Indicates a two-way parameter.
AdParamReturnValue	Indicates a return value.

Direction Property Remarks

Use the **Direction** property to specify how a parameter is passed to or from a procedure. The **Direction** property is read/write; this allows you to work with providers that do not return this information, or to set this information when you do not want ADO to make an extra call to the provider to retrieve parameter information.

Not all providers can determine the direction of parameters in their stored procedures. In these cases, you must set the **Direction** property prior to executing the query.

ADO Parameter Object Name Property

The name of an object.

Name Property Return Values (ADO Parameter Object)

Sets or returns a **String** value. The value is read/write on a **Parameter** object.

Name Property Remarks (ADO Parameter Object)

Use the **Name** property to assign a name to or retrieve the name of a **Parameter** object.

For **Parameter** objects not yet appended to the ADO Parameters Collection, the **Name** property is read/write. For appended **Parameter** objects and all other objects, the **Name** property is read-only. Names do not have to be unique within a collection.

ADO Parameter Object NumericScale Property

The scale of **Numeric** values in a **Parameter** object.

NumericScale Property Return Values

Sets or returns a **Byte** value, indicating the number of decimal places to which numeric values will be resolved.

NumericScale Property Remarks

Use the **NumericScale** property to determine how many digits to the right of the decimal point will be used to represent values for a numeric **Parameter** object.

For **Parameter** objects, the **NumericScale** property is read/write.

ADO Parameter Object Precision Property

The degree of precision for **Numeric** values in a **Parameter** object.

Precision Property Return Values (ADO Parameter Object)

Sets or returns a **Byte** value, indicating the maximum total number of digits used to represent values. The value is read/write on a **Parameter** object.

Precision Property Remarks (ADO Parameter Object)

Use the **Precision** property to determine the maximum number of digits used to represent values for a numeric **Parameter** object.

ADO Parameter Object Size Property

The maximum size, in bytes or characters, of a **Parameter** object.

Size Property Return Values (ADO Parameter Object)

Sets or returns a **Long** value that indicates the maximum size in bytes or characters of a value in a **Parameter** object.

Size Property Remarks (ADO Parameter Object)

Use the **Size** property to determine the maximum size for values written to or read from the ADO Parameter Object Value Property of a **Parameter** object. The **Size** property is read/write. If you specify a variable-length data type for a **Parameter** object, you must set the object's **Size** property before appending it to the ADO Parameters Collection; otherwise an error occurs. If you have already appended the **Parameter** object to the **Parameters** collection of an ADO Command Object and you change its type to a variable-length data type, you must set the **Parameter** object's **Size** property before executing the **Command** object; otherwise an error occurs.

If you use the ADO Collections Refresh Method to obtain parameter information from the provider and it returns one or more variable-length data type **Parameter** objects, ADO may allocate memory for the parameters based on their maximum potential size, which could cause an error during execution. To prevent an error, you should explicitly set the **Size** property for these parameters before executing the command.

Size Property Example (ADO Parameter Object)

See **ActiveConnection** property example.

ADO Parameter Object Type Property

The operational type or data type of a **Parameter** object.

Type Property Return Values (ADO Parameter Object)

Sets or returns one of the following **DataTypeEnum** values. The corresponding OLE DB type indicators are as follows:

Constant	Description
adArray	OR'd together with another type to indicate that the data is a

	safe-array of that type (DBTYPE_ARRAY).
adBigInt	An 8-byte signed integer (DBTYPE_I8).
adBinary	A binary value (DBTYPE_BYTES).
adBoolean	A Boolean value (DBTYPE_BOOL).
adByRef	Or'd together with another type to indicate that the data is a pointer to data of the other type (DBTYPE_BYREF).
adBSTR	A null-terminated character string (Unicode) (DBTYPE_BSTR).
adChar	A String value (DBTYPE_STR).
adCurrency	A currency value (DBTYPE_CY). Currency is a fixed-point number with 4 digits to the right of the decimal point. It is stored in an 8-byte signed integer scaled by 10,000.
adDate	A date value (DBTYPE_DATE). A date is stored as a Double, the whole part of which is the number of days since December 30, 1899, and the fractional part of which is the fraction of a day.
adDBDate	A date value (yyyymmdd) (DBTYPE_DBDATE).
adDBTime	A time value (hhmmss) (DBTYPE_DBTIME).
adDBTimeStamp	A date-time stamp (yyyymmddhhmmss plus a fraction in billionths) (DBTYPE_DBTIMESTAMP).
adDecimal	An exact numeric value with a fixed precision and scale (DBTYPE_DECIMAL).
adDouble	A double-precision floating point value (DBTYPE_R8).
adEmpty	No value was specified (DBTYPE_EMPTY).
adError	A 32-bit error code (DBTYPE_ERROR).
adGUID	A globally unique identifier (GUID) (DBTYPE_GUID).
adIDispatch	A pointer to an **IDispatch** interface on an OLE object (DBTYPE_IDISPATCH).
adInteger	A 4-byte signed integer (DBTYPE_I4).
adIUnknown	A pointer to an **IUnknown** interface on an OLE object (DBTYPE_IUNKNOWN).
adLongVarBinary	A long binary value.
adLongVarChar	A long String value.
adLongVarWChar	A long null-terminated string value.
adNumeric	An exact numeric value with a fixed precision and scale (DBTYPE_NUMERIC).

adSingle	A single-precision floating point value (DBTYPE_R4).
adSmallInt	A 2-byte signed integer (DBTYPE_I2).
adTinyInt	A 1-byte signed integer (DBTYPE_I1).
adUnsignedBigInt	An 8-byte unsigned integer (DBTYPE_UI8).
adUnsignedInt	A 4-byte unsigned integer (DBTYPE_UI4).
adUnsignedSmallInt	A 2-byte unsigned integer (DBTYPE_UI2).
adUnsignedTinyInt	A 1-byte unsigned integer (DBTYPE_UI1).
adUserDefined	A user-defined variable (DBTYPE_UDT).
adVarBinary	A binary value.
adVarChar	A String value.
adVariant	An Automation **Variant** (DBTYPE_VARIANT).
adVector	OR'd together with another type to indicate that the data is a DBVECTOR structure, as defined by OLE DB, that contains a count of elements and a pointer to data of the other type (DBTYPE_VECTOR).
adVarWChar	A null-terminated Unicode character string.
adWChar	A null-terminated Unicode character string (DBTYPE_WSTR).

Type Property Remarks (ADO Parameter Object)

For **Parameter** objects, the **Type** property is read/write.

ADO Parameter Object Value Property

Indicates the value assigned to a **Parameter** object.

Value Property Return Values (ADO Parameter Object)

Sets or returns a **Variant** value. Default value depends on the ADO Parameter Object Type Property.

Value Property Remarks (ADO Parameter Object)

Use the **Value** property to set or return parameter values with **Parameter** objects.

ADO allows setting and returning long binary data with the **Value** property.

ADO Property Object

ADO Property Object

The ADO **Property** object represents a dynamic characteristic of an ADO object that is defined by the provider. *This object is not currently supported on UNIX.[0]*

ADO Property Object Properties

ADO Property Object Attributes Property	One or more characteristics of a property.
ADO Property Object Name Property	The name of the property.
ADO Property Object Type Property	The operational or data type of the property.
ADO Property Object Value Property	The value assigned to the property.

ADO Property Object Remarks

ADO objects have two types of properties: built-in and dynamic. Built-in properties are those properties implemented in ADO and immediately available to any new object, using the familiar `MyObject.Property` syntax.

Built-in properties do not appear as **Property** objects in an object's ADO Properties Collection, so while you can change their values, you cannot modify their characteristics or delete them.

Dynamic properties are defined by the underlying data provider, and appear in the **Properties** collection for the appropriate ADO object. For example, a property specific to the provider may indicate if an ADO Recordset Object supports transactions or updating. These additional properties will appear as **Property** objects in that **Recordset** object's **Properties** collection. Dynamic properties can be referenced only through the collection, using the `MyObject.Properties(0)` or `MyObject.Properties("Name")` syntax.

A dynamic **Property** object has four built-in properties:

- The ADO Property Object Name Property is a string that identifies the property.
- The ADO Property Object Type Property is an integer that specifies the property data type.
- The ADO Property Object Value Property is a variant that contains the property setting.
- The ADO Property Object Attributes Property is a long value that indicates characteristics of the property specific to the provider.

ADO Property Object Properties

ADO Property Object Attributes Property

One or more characteristics of an object.

Attributes Property Return Values (ADO Property Object)

Sets or returns a **Long** value.

Attributes Property Property (ADO Property Object)

For a **Property** object, the **Attributes** property is read-only, and its value can be the sum of any one or more of these **PropertyAttributesEnum** values:

Value	Description
adPropNotSupported	The property is not supported by the provider.
adPropRequired	The user must specify a value for this property before the data source is initialized.
adPropOptional	The user does not need to specify a value for this property before the data source is initialized.
adPropRead	The user can read the property.
adPropWrite	The user can set the property.

Attributes Property Remarks (ADO Property Object)

Use the **Attributes** property to set or return characteristics of **Property** objects.

When you set multiple attributes, you can sum the appropriate constants. If you set the property value to a sum including incompatible constants, an error occurs.

Attributes Property Examples (ADO Property Object)

This Visual Basic example displays the value of the **Attributes** property for **Property** objects. It uses the ADO Property Object Name Property to display the name of each **Property** object.

```
Public Sub AttributesX

Dim cnn1 As ADODB.Connection

Dim rstEmployees As ADODB.Recordset

Dim fldLoop As ADODB.Field

Dim proLoop As ADODB.Property

Dim strCnn As String

' Open connection and recordset.

strCnn = "driver={SQL Server};server=srv;" & _

"uid=sa;pwd=;database=pubs"

Set cnn1 = New ADODB.Connection

cnn1.Open strCnn

Set rstEmployees = New ADODB.Recordset

rstEmployees.Open "employee", cnn1, , ,

adCmdTable

' Display attributes of the Employee table properties.

Debug.Print "Property attributes:"
```

```
For Each proLoop In rstEmployees.Properties
Debug.Print " " & proLoop.Name & " = " & _
proLoop.Attributes
Next proLoop
rstEmployees.Close
cnn1.Close
End Sub
```

ADO Property Object Name Property

The name of an object.

Name Property Return Values (ADO Property Object)

Sets or returns a **String** value. The value is read-only on a **Property** object.

Name Property Remarks (ADO Property Object)

Use the **Name** property to assign a name to or retrieve the name of a **Property** object.

You can retrieve the **Name** property of an object by an ordinal reference, after which the object can be referred to directly by name. For example, if `rstMain.Properties(20).Name` yields `Updatability`, you can subsequently refer to this property as `rstMain.Properties("Updatability").`

Name Property Examples (ADO Property Object)

See the ADO Property Object Attributes Property example.

ADO Property Object Type Property

The operational type or data type of a **Property** object.

Type Property Return Values (ADO Property Object)

Sets or returns one of the following **DataTypeEnum** values. The corresponding OLE DB type indicators are as follows:

Constant	Description
adArray	Or'd together with another type to indicate that the data is a safe-array of that type (DBTYPE_ARRAY).
adBigInt	An 8-byte signed integer (DBTYPE_I8).
adBinary	A binary value (DBTYPE_BYTES).
adBoolean	A Boolean value (DBTYPE_BOOL).
adByRef	Or'd together with another type to indicate that the data is a

	pointer to data of the other type (DBTYPE_BYREF).
adBSTR	A null-terminated character string (Unicode) (DBTYPE_BSTR).
adChar	A String value (DBTYPE_STR).
adCurrency	A currency value (DBTYPE_CY). Currency is a fixed-point number with 4 digits to the right of the decimal point. It is stored in an 8-byte signed integer scaled by 10,000.
adDate	A date value (DBTYPE_DATE). A date is stored as a Double, the whole part of which is the number of days since December 30, 1899, and the fractional part of which is the fraction of a day.
adDBDate	A date value (yyyymmdd) (DBTYPE_DBDATE).
adDBTime	A time value (hhmmss) (DBTYPE_DBTIME).
adDBTimeStamp	A date-time stamp (yyyymmddhhmmss plus a fraction in billionths) (DBTYPE_DBTIMESTAMP).
adDecimal	An exact numeric value with a fixed precision and scale (DBTYPE_DECIMAL).
adDouble	A double-precision floating point value (DBTYPE_R8).
adEmpty	No value was specified (DBTYPE_EMPTY).
adError	A 32-bit error code (DBTYPE_ERROR).
adGUID	A globally unique identifier (GUID) (DBTYPE_GUID).
adIDispatch	A pointer to an **IDispatch** interface on an OLE object (DBTYPE_IDISPATCH).
adInteger	A 4-byte signed integer (DBTYPE_I4).
adIUnknown	A pointer to an **IUnknown** interface on an OLE object (DBTYPE_IUNKNOWN).
adLongVarBinary	A long binary value.
adLongVarChar	A long String value.
adLongVarWChar	A long null-terminated string value.
adNumeric	An exact numeric value with a fixed precision and scale (DBTYPE_NUMERIC).
adSingle	A single-precision floating point value (DBTYPE_R4).
adSmallInt	A 2-byte signed integer (DBTYPE_I2).
adTinyInt	A 1-byte signed integer (DBTYPE_I1).
adUnsignedBigInt	An 8-byte unsigned integer (DBTYPE_UI8).
adUnsignedInt	A 4-byte unsigned integer (DBTYPE_UI4).
adUnsignedSmallInt	A 2-byte unsigned integer (DBTYPE_UI2).

adUnsignedTinyInt	A 1-byte unsigned integer (DBTYPE_UI1).
adUserDefined	A user-defined variable (DBTYPE_UDT).
adVarBinary	A binary value.
adVarChar	A String value.
adVariant	An Automation **Variant** (DBTYPE_VARIANT).
adVector	OR'd together with another type to indicate that the data is a DBVECTOR structure, as defined by OLE DB, that contains a count of elements and a pointer to data of the other type (DBTYPE_VECTOR).
adVarWChar	A null-terminated Unicode character string.
adWChar	A null-terminated Unicode character string (DBTYPE_WSTR).

Type Property Remarks (ADO Property Object)

The **Type** property is read-only.

ADO Property Object Value Property

Indicates the value assigned to a **Property** object.

Value Property Return Values (ADO Property Object)

Sets or returns a **Variant** value. Default value depends on the ADO Property Object Type Property.

Value Property Remarks (ADO Property Object)

Use the **Value** property to set or return property settings with **Property** objects.

Value Property Example (ADO Property Object)

This Visual Basic example demonstrates the **Value** property with **Field** and **Property** objects by displaying field and property values for the ***Employees*** table.

```
Public Sub ValueX()
Dim rstEmployees As ADODB.Recordset
Dim fldLoop As ADODB.Field
Dim prpLoop As ADODB.Property
Dim strCnn As String
' Open recordset with data from Employee table.
strCnn = "driver={SQL Server};server=srv;" & _
"uid=sa;pwd=;database=pubs"
Set rstEmployees = New ADODB.Recordset
```

```
rstEmployees.Open "employee", strCnn, , , adCmdTable
Debug.Print "Field values in rstEmployees"
' Enumerate the Fields collection of the Employees
' table.
For Each fldLoop In rstEmployees.Fields
` Because Value is the default property of a
` Field object, the use of the actual keyword
` here is optional.
Debug.Print " " & fldLoop.Name & " = " &
fldLoop.Value
Next fldLoop
Debug.Print "Property values in rstEmployees"
' Enumerate the Properties collection of the
' Recordset object.
For Each prpLoop In rstEmployees.Properties
' Because Value is the default property of a
' Property object, the use of the actual keyword
' here is optional.
Debug.Print " " & prpLoop.Name & " = " &
prpLoop.Value
Next prpLoop
rstEmployees.Close
End Sub
```

ADO Recordset Object

ADO Recordset Object

The **Recordset** object represents the entire set of records from a database table or the results of an executed command.

ADO Recordset Object Collections

ADO Fields Collection	All the stored **Field** objects of a **Recordset** object.
ADO Properties Collection	All the **Property** objects for a specific instance of a **Recordset** object. *This collection is not currently*

supported on UNIX.

ADO Recordset Object Methods

ADO Recordset Object AddNew Method	Creates a new record for an updatable **Recordset** object.
ADO Recordset Object CancelBatch Method	Cancels a pending batch update. *This method is not currently supported on UNIX.*
ADO Recordset Object CancelUpdate Method	Cancels any changes made to the current record prior to calling the **Update** method.
ADO Recordset Object Clone Method	Creates a new **Recordset** object from an existing **Recordset** object. *This method is not currently supported on UNIX.*
ADO Recordset Object Close Method	Closes an open **Recordset** object and any dependent objects.
ADO Recordset Object Delete Method	Deletes the current record or group of records from a **Recordset** object.
ADO Recordset Object GetRows Method	Retrieves multiple rows from a **Recordset** object into an array.
ADO Recordset Object Move Method	Moves the position of the current record in a **Recordset** object.
ADO Recordset Object MoveFirst Method	Moves to the first record in a **Recordset** object and makes that record the current record.
ADO Recordset Object MoveLast Method	Moves to the last record in a **Recordset** object and makes that record the current record.
ADO Recordset Object MoveNext Method	Moves to the next record in a **Recordset** object and makes that record the current record.
ADO Recordset Object MovePrevious Method	Moves to the previous record in a **Recordset** object and makes that record the current record.
ADO Recordset Object NextRecordset Method	Clears the current **Recordset** object and returns the next recordset by advancing through a series of commands. *This method is not currently supported on UNIX.*
ADO Recordset Object Open Method	Opens a cursor.
ADO Recordset Object Requery Method	Updates the data in a recordset by re-executing the query on which the object is based.
ADO Recordset Object Resync Method	Refreshes the data in the **Recordset** object from the underlying database.
ADO Recordset Object Supports	Determines whether a specified **Recordset** object

Method	supports a particular type of functionality.
ADO Recordset Object Update Method	Saves any changes you make to the current record of a **Recordset** object.
ADO Recordset Object UpdateBatch Method	Writes all pending batch updates. *This method is not currently supported on UNIX.*

ADO Recordset Object Properties

ADO Recordset Object AbsolutePage Property	The page in which the current record resides.
ADO Recordset Object AbsolutePosition Property	The ordinal position of a **Recordset** object's current position.
ADO Recordset Object ActiveConnection Property	The **Connection** object to which the **Recordset** object currently belongs.
ADO Recordset Object BOF, EOF Properties	If **True**, the current record position is before the first record in a **Recordset** object.
ADO Recordset Object Bookmark Property	A value that uniquely identifies the current record in a **Recordset** object. Setting the **Bookmark** property to a valid bookmark changes the current record.
ADO Recordset Object CacheSize Property	The number of records from a **Recordset** object that are cached locally in memory. *This property is not currently supported on UNIX.*
ADO Recordset Object CursorLocation Property	The location of the cursor engine.
ADO Recordset Object CursorType Property	The type of cursor used in a **Recordset** object.
ADO Recordset Object EditMode Property	The editing status of the current record.
ADO Recordset Object BOF, EOF Properties	**True** if the record position is after the last record in a **Recordset** object.
ADO Recordset Object Filter Property	A filter for data in a **Recordset** object.
ADO Recordset Object LockType Property	The type of locks placed on records during editing.
ADO Recordset Object MarshalOptions Property	Which records are to be marshaled back to the server.
ADO Recordset Object MaxRecords Property	The maximum number of records to return to a **Recordset** object from a query.
ADO Recordset Object PageCount	The number of pages of data the **Recordset** object

Property	contains.
ADO Recordset Object PageSize Property	The number of records that make up one page in the **Recordset** object.
ADO Recordset Object RecordCount Property	The current number of records in a **Recordset** object.
ADO Recordset Object Source Property	The source for the data in a **Recordset** object.
ADO Recordset Object State Property	Describes the current state of the **Recordset** object.
ADO Recordset Object Status Property	The status of the current record with respect to batch updates or other bulk operations.

ADO Recordset Object Remarks

Use **Recordset** objects to manipulate data from a provider. In ADO, data is almost entirely manipulated using **Recordset** objects. All **Recordset** objects are constructed using records (rows) and fields (columns). Depending on the functionality supported by the provider, some **Recordset** methods or properties may not be available.

Recordset objects can also be run remotely. For example, in a Web-based application, you can open a **Recordset** on the client, using the progID "ADOR." The Remote Data Service provides a mechanism for local data caching and local cursor movement in remote recordset data. A client-side recordset can be used in the same way as a server-side recordset, and supports almost all of the **Recordset** object's normal methods and properties. **Recordset** methods and properties that are not supported on a client-side recordset, or that behave differently, are noted in the topics for those properties and methods.

There are four different cursor types defined in ADO:

Cursor	Description
Dynamic	Allows you to view additions, changes and deletions by other users, and allows all types of movement through the recordset that don’t rely on bookmarks; allows bookmarks if the provider supports them.
Keyset	Behaves like a dynamic cursor, except that it prevents you from seeing records that other users add, and prevents access to records that other users delete. Data change by other users will still be visible. It always supports bookmarks and therefore allows all types of movement through the recordset.
Static	Provides a static copy of a set of records for you to use to find data or generate reports. Always allows bookmarks and therefore allows all types of movement through the recordset. Additions, changes, or deletions by other users will not be visible. This is the only type of cursor allowed when you open a client-side (ADOR) **Recordset** object.
Forward-only	Behaves identically to a dynamic cursor except that it allows you to scroll only forward through records. This improves performance in situations

where you need to make only a single pass through a recordset.

Set the ADO Recordset Object CursorType Property prior to opening the recordset to choose the cursor type, or pass a *CursorType* argument with the ADO Recordset Object Open Method. Some providers don't support all cursor types. Check the documentation for the provider. If you don't specify a cursor type, ADO opens a forward-only cursor by default.

When used with some providers (such as the Microsoft ODBC Provider for OLE DB in conjunction with Microsoft SQL Server), you can create **Recordset** objects independently of a previously defined ADO Connection Object by passing a connection string with the **Open** method. ADO still creates a **Connection** object, but it doesn't assign that object to an object variable. However, if you are opening multiple **Recordset** objects over the same connection, you should explicitly create and open a **Connection** object; this assigns the **Connection** object to an object variable. If you do not use this object variable when opening your **Recordset** objects, ADO creates a new **Connection** object for each new recordset, even if you pass the same connection string.

You can create as many **Recordset** objects as needed.

When you open a recordset, the current record is positioned to the first record (if any) and the ADO Recordset Object BOF, EOF Properties are set to **False**. If there are no records, the **BOF** and **EOF** property settings are **True**.

Use the ADO Recordset Object MoveFirst, MoveLast, MoveNext, MovePrevious Methods, as well as the ADO Recordset Object Move Method, and the **AbsolutePosition, AbsolutePage,** and ADO Recordset Object Filter Property properties to reposition the current record, assuming the provider supports the relevant functionality. Forward-only **Recordset** objects support only the **MoveNext** method. When you use the **Move** methods to visit each record (or enumerate the recordset), you can use the **BOF** and **EOF** properties to see if you've moved beyond the beginning or end of the recordset.

Recordset objects may support two types of updating: immediate and batched. In immediate updating, all changes to data are written immediately to the underlying data source once you call the ADO Recordset Object Update Method. You can also pass arrays of values as parameters with the ADO Recordset Object AddNew Method and **Update** methods and simultaneously update several fields in a record.

If a provider supports batch updating, you can have the provider cache changes to more than one record and then transmit them in a single call to the database with the ADO Recordset Object UpdateBatch Method. This applies to changes made with the **AddNew**, **Update**, and ADO Recordset Object Delete Method methods. After you call the **UpdateBatch** method, you can use the ADO Recordset Object Status Property to check for any data conflicts in order to resolve them. Batch updating is not currently supported on UNIX.

Note

To execute a query without using an ADO Command Object, pass a query string to the ADO Recordset Object Open Method of a **Recordset** object. However, a **Command** object is required when you want to retain the command text and re-execute it, or use query parameters.

ADO Recordset Object Methods

ADO Recordset Object AddNew Method

Creates a new record for an updateable **Recordset** object.

AddNew Method Syntax

```
recordset.AddNew Fields, Values
```

AddNew Method Parameters

Fields

An optional single name or an array of names or ordinal positions of the fields in the new record.

Values

An optional single value or an array of values for the fields in the new record. If **Fields** is an array, **Values** must also be an array with the same number of members; otherwise, an error occurs. The order of field names must match the order of field values in each array.

AddNew Method Remarks

Use the **AddNew** method to create and initialize a new record. Use the ADO Recordset Object Supports Method with **adAddNew** to verify whether you can add records to the current **Recordset** object.

After you call the **AddNew** method, the new record becomes the current record and remains current after you call the ADO Recordset Object Update Method. If the **Recordset** object does not support bookmarks, you may not be able to access the new record once you move to another record. Depending on your cursor type, you may need to call the ADO Recordset Object Requery Method to make the new record accessible.

If you call **AddNew** while editing the current record or while adding a new record, ADO calls the **Update** method to save any changes and then creates the new record.

The behavior of the **AddNew** method depends on the updating mode of the **Recordset** object and whether or not you pass the *Fields* and *Values* arguments.

In *immediate update mode* (the provider writes changes to the underlying data source once you call the **Update** method), calling the **AddNew** method without arguments sets the ADO Recordset Object EditMode Property to **adEditAdd**. The provider caches any field value changes locally. Calling the **Update** method posts the new record to the database and resets the **EditMode** property to **adEditNone**. If you pass the *Fields* and *Values* arguments, ADO immediately posts the new record to the database (no **Update** call is necessary); the **EditMode** property value does not change (**adEditNone**).

In *batch update mode* (the provider caches multiple changes and writes them to the underlying data source only when you call the **UpdateBatch** method), calling the **AddNew** method without arguments sets the **EditMode** property to **adEditAdd**. The provider caches any field value changes locally. Calling the **Update** method adds the new record to the current recordset and resets the **EditMode** property to **adEditNone**, but the provider does not post the changes to the

underlying database until you call the ADO Recordset Object UpdateBatch Method. If you pass the *Fields* and *Values* arguments, ADO sends the new record to the provider for storage in a cache; you need to call the **UpdateBatch** method to post the new record to the underlying database. Batch updating is not currently supported on UNIX.

ADO Recordset Object CancelBatch Method

Cancels a pending batch update. *This method is not currently supported on UNIX.*

CancelBatch Method Syntax

```
recordset.CancelBatch AffectRecords
```

CancelBatch Method Parameters

AffectRecords

An optional **AffectEnum** value that determines how many records the **CancelBatch** method will affect. It can be one of the following constants:

Constant	Description
adAffectCurrent	Cancels pending updates only for the current record.
adAffectGroup	Cancels pending updates for records that satisfy the current ADO Recordset Object Filter Property setting. You must set the **Filter** property to one of the valid predefined constants in order to use this option.
adAffectAll	Default. Cancels pending updates for all the records in the **Recordset** object, including any hidden by the current **Filter** property setting.

CancelBatch Method Remarks

Use the **CancelBatch** method to cancel any pending updates in a recordset in batch update mode. If the recordset is in immediate update mode, calling **CancelBatch** without **adAffectCurrent** generates an error.

If you are editing the current record or are adding a new record when you call **CancelBatch**, ADO first calls the ADO Recordset Object CancelUpdate Method to cancel any cached changes; after that, all pending changes in the recordset are canceled.

It's possible that the current record will be indeterminable after a **CancelBatch** call, especially if you were in the process of adding a new record. For this reason, it is prudent to set the current record position to a known location in the recordset after the **CancelBatch** call. For example, call the ADO Recordset Object MoveFirst, MoveLast, MoveNext, and MovePrevious Methods.

If the attempt to cancel the pending updates fails because of a conflict with the underlying data (for example, a record has been deleted by another user), the provider returns warnings to the ADO Errors Collection but does not halt program execution. A run-time error occurs only if there are conflicts on all the requested records. Use the **Filter** property (**adFilterAffectedRecords**) and the ADO Recordset Object Status Property to locate records with conflicts.

CancelBatch Method Examples

This Visual Basic example demonstrates the ADO Recordset Object UpdateBatch Method in conjunction with the **CancelBatch** method.

```
Public Sub UpdateBatchX()
Dim rstTitles As ADODB.Recordset
Dim strCnn As String
Dim strTitle As String
Dim strMessage As String
` Assign connection string to variable.
strCnn = "driver={SQL Server};server=srv;" & _
"uid=sa;pwd=;database=pubs"
Set rstTitles = New ADODB.Recordset
rstTitles.CursorType = adOpenKeyset
rstTitles.LockType = adLockBatchOptimistic
rstTitles.Open "titles", strCnn, , , adCmdTable
rstTitles.MoveFirst
` Loop through recordset and ask user if she wants
` to change the type for a specified title.
Do Until rstTitles.EOF
If Trim(rstTitles!Type) = "psychology" Then
strTitle = rstTitles!Title
strMessage = "Title: " & strTitle & vbCr & _
"Change type to self help?"
If MsgBox(strMessage, vbYesNo) = vbYes Then
rstTitles!Type = "self_help"
End If
End If
rstTitles.MoveNext
Loop
` Ask if the user wants to commit to all the
` changes made above.
If MsgBox("Save all changes?", vbYesNo) = vbYes Then
rstTitles.UpdateBatch
```

```
Else
rstTitles.CancelBatch
End If
` Print current data in recordset.
rstTitles.Requery
rstTitles.MoveFirst
Do While Not rstTitles.EOF
Debug.Print rstTitles!Title & " - " & rstTitles!Type
rstTitles.MoveNext
Loop
` Restore original values because this is a demonstration.
rstTitles.MoveFirst
Do Until rstTitles.EOF
If Trim(rstTitles!Type) = "self_help" Then
rstTitles!Type = "psychology"
End If
rstTitles.MoveNext
Loop
rstTitles.UpdateBatch
rstTitles.Close
End Sub
```

ADO Recordset Object CancelUpdate Method

Cancels any changes made to the current record or to a new record prior to calling the **Update** method.

CancelUpdate Method Syntax

recordset.**CancelUpdate**

CancelUpdate Method Remarks

Use the **CancelUpdate** method to cancel any changes made to the current record or to discard a newly added record.

Note

You cannot undo changes to the current record or to a new record after you call the ADO Recordset Object Update Method unless the changes are either part of a transaction that you can roll back with the **RollbackTrans** method or part of a batch update that you can cancel with the ADO Recordset Object CancelBatch Method.

If you are adding a new record when you call the **CancelUpdate** method, the record that was current prior to the ADO Recordset Object AddNew Method call becomes the current record again. If you have not changed the current record or added a new record, calling the **CancelUpdate** method generates an error.

CancelUpdate Method Examples

These Visual Basic examples demonstrate the ADO Recordset Object Update Method in conjunction with the **CancelUpdate** method.

```
Public Sub UpdateX()

Dim rstEmployees As ADODB.Recordset

Dim strOldFirst As String

Dim strOldLast As String

Dim strMessage As String

` Open recordset with names from Employee table.

strCnn = "driver={SQL Server};server=srv;" & _

"uid=sa;pwd=;database=pubs"

Set rstEmployees = New ADODB.Recordset

rstEmployees.CursorType = adOpenKeyset

rstEmployees.LockType = adLockOptimistic

rstEmployees.Open "SELECT fname, lname " & _

"FROM Employee ORDER BY lname", strCnn, , , adCmdText

` Store original data.

strOldFirst = rstEmployees!fname

strOldLast = rstEmployees!lname

` Change data in edit buffer.

rstEmployees!fname = "Linda"

rstEmployees!lname = "Kobara"

` Show contents of buffer and get user input.

strMessage = "Edit in progress:" & vbCr & _

" Original data = " & strOldFirst & " " & _
```

```
strOldLast & vbCr & " Data in buffer = " & _
rstEmployees!fname & " " & rstEmployees!lname & vbCr & vbCr & _
"Use Update to replace the original data with " & _
"the buffered data in the Recordset?"
If MsgBox(strMessage, vbYesNo) = vbYes Then
rstEmployees.Update
Else
rstEmployees.CancelUpdate
End If
` Show the resulting data.
MsgBox "Data in recordset = " & rstEmployees!fname & " " & _
rstEmployees!lname
` Restore original data because this is a demonstration.
If Not (strOldFirst = rstEmployees!fname And _
strOldLast = rstEmployees!lname) Then
rstEmployees!fname = strOldFirst
rstEmployees!lname = strOldLast
rstEmployees.Update
End If
rstEmployees.Close
End Sub
```

This example demonstrates the **Update** method in conjunction with the **AddNew** method.

```
Public Sub UpdateX2()
Dim cnn1 As ADODB.Connection
Dim rstEmployees As ADODB.Recordset
Dim strEmpID As String
Dim strOldFirst As String
Dim strOldLast As String
Dim strMessage As String
` Open a connection.
Set cnn1 = New ADODB.Connection
strCnn = "driver={SQL Server};server=srv;" & _
```

```
"uid=sa;pwd=;database=pubs"
cnn1.Open strCnn
` Open recordset with data from Employee table.
Set rstEmployees = New ADODB.Recordset
rstEmployees.CursorType = adOpenKeyset
rstEmployees.LockType = adLockOptimistic
rstEmployees.Open "employee", cnn1, , , adCmdTable
rstEmployees.AddNew
strEmpID = "B-S55555M"
rstEmployees!emp_id = strEmpID
rstEmployees!fname = "Bill"
rstEmployees!lname = "Sornsin"
` Show contents of buffer and get user input.
strMessage = "AddNew in progress:" & vbCr & _
"Data in buffer = " & rstEmployees!emp_id & ", " & _
rstEmployees!fname & " " & rstEmployees!lname & vbCr & vbCr & _
"Use Update to save buffer to recordset?"
If MsgBox(strMessage, vbYesNoCancel) = vbYes Then
rstEmployees.Update
` Go to the new record and show the resulting data.
MsgBox "Data in recordset = " & rstEmployees!emp_id & ", " & _
rstEmployees!fname & " " & rstEmployees!lname
Else
rstEmployees.CancelUpdate
MsgBox "No new record added."
End If
` Delete new data because this is a demonstration.
cnn1.Execute "DELETE FROM employee WHERE emp_id = '" & strEmpID &
"'"
rstEmployees.Close
End Sub
```

ADO Recordset Object Clone Method

Creates a duplicate **Recordset** object from an existing **Recordset** object. *This method is not currently supported on UNIX.*

Clone Method Syntax

```
Set rstDuplicate = rstOriginal.Clone ()
```

Clone Method Parameters

rstDuplicate

An object variable identifying the duplicate **Recordset** object you're creating.

rstOriginal

An object variable identifying the **Recordset** object you want to duplicate.

Clone Method Remarks

Use the **Clone** method to create multiple, duplicate **Recordset** objects, particularly if you want to be able to maintain more than one current record in a given set of records. Using the **Clone** method is more efficient than creating and opening a new **Recordset** object with the same definition as the original.

The current record of a newly created clone is set to the first record.

Changes you make to one **Recordset** object are visible in all of its clones regardless of cursor type. However, once you execute the ADO Recordset Object Requery Method on the original **Recordset**, the clones will no longer be synchronized to the original.

Closing the original recordset does not close its copies; closing a copy does not close the original or any of the other copies.

You can only clone a **Recordset** object that supports bookmarks. Bookmark values are interchangeable; that is, a bookmark reference from one **Recordset** object refers to the same record in any of its clones.

Clone Method Return Values

Returns a **Recordset** object reference.

ADO Recordset Object Close Method

Closes an open object and any dependent objects.

Close Method Syntax

```
object.Close
```

Close Method Remarks

Use the **Close** method to close a **Recordset** object to free any associated system resources. Closing an object does not remove it from memory; you may change its property settings and open it again later. To completely eliminate an object from memory, set the object variable to *Nothing*.

Using the **Close** method to close a **Recordset** object releases the associated data and any exclusive access you may have had to the data through this particular **Recordset** object. You can later call the ADO Recordset Object Open Method to reopen the **Recordset** with the same or modified attributes. While the **Recordset** object is closed, calling any methods that require a live cursor generates an error.

If an edit is in progress while in immediate update mode, calling the **Close** method generates an error; call the ADO Recordset Object Update Method or ADO Recordset Object CancelUpdate Method first. If you close the **Recordset** object during batch updating, all changes since the last ADO Recordset Object UpdateBatch Method call are lost.

If you use the **Clone** method to create copies of an open **Recordset** object, closing the original or a clone does not affect any of the other copies.

Close Method Examples

This VBScript example uses the **Open** and **Close** methods on both **Recordset** and **Connection** objects that have been opened.

```
<!-- #Include file="ADOVBS.INC" -->
<HTML><HEAD>
<TITLE>ADO 1.5 Open Method</TITLE>
</HEAD><BODY>
<FONT FACE="MS SANS SERIF" SIZE=2>
<Center><H3>ADO Open Method</H3>
<TABLE WIDTH=600 BORDER=0>
<TD VALIGN=TOP ALIGN=LEFT COLSPAN=3><FONT SIZE=2>
<!--- ADO Connection used to create 2 recordsets-->
<%
Set OBJdbConnection = Server.CreateObject("ADODB.Connection")
OBJdbConnection.Open "AdvWorks"
SQLQuery = "SELECT * FROM Customers"
'First Recordset RSCustomerList
Set RSCustomerList = OBJdbConnection.Execute(SQLQuery)
'Second Recordset RsProductist
Set RsProductList = Server.CreateObject("ADODB.Recordset")
RsProductList.CursorType = adOpenDynamic
RsProductList.LockType = adLockOptimistic
RsProductList.Open "Products", OBJdbConnection
%>
```

```
<TABLE COLSPAN=8 CELLPADDING=5 BORDER=0>
<!-- BEGIN column header row for Customer Table-->
<TR><TD ALIGN=CENTER BGCOLOR="#008080">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>Company
Name</FONT></TD>
<TD ALIGN=CENTER BGCOLOR="#008080">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>Contact
Name</FONT></TD>
<TD ALIGN=CENTER WIDTH=150 BGCOLOR="#008080">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>E-mail
address</FONT></TD>
<TD ALIGN=CENTER BGCOLOR="#008080">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>City</FONT></TD>
<TD ALIGN=CENTER BGCOLOR="#008080">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff"
SIZE=1>State/Province</FONT></TD></TR>
<!--Display ADO Data from Customer Table-->
<% Do While Not RScustomerList.EOF %>
<TR><TD BGCOLOR="f7efde" ALIGN=CENTER>
<FONT STYLE="ARIAL NARROW" SIZE=1>
<%= RSCustomerList("CompanyName")%>
</FONT></TD>
<TD BGCOLOR="f7efde" ALIGN=CENTER>
<FONT STYLE="ARIAL NARROW" SIZE=1>
<%= RScustomerList("ContactLastName") & ", " %>
<%= RScustomerList("ContactFirstName") %>
</FONT></TD>
<TD BGCOLOR="f7efde" ALIGN=CENTER>
<FONT STYLE="ARIAL NARROW" SIZE=1>
<%= RScustomerList("ContactLastName")%>
</FONT></TD>
<TD BGCOLOR="f7efde" ALIGN=CENTER>
<FONT STYLE="ARIAL NARROW" SIZE=1>
```

```
<%= RScustomerList("City")%>
</FONT></TD>
<TD BGCOLOR="f7efde" ALIGN=CENTER>
<FONT STYLE="ARIAL NARROW" SIZE=1>
<%= RScustomerList("StateOrProvince")%>
</FONT></TD></TR>
<!-Next Row = Record Loop and add to html table-->
<%
RScustomerList.MoveNext
Loop
RScustomerList.Close
OBJdbConnection.Close
%>
</TABLE>
<HR>
<TABLE COLSPAN=8 CELLPADDING=5 BORDER=0>
<!-- BEGIN column header row for Product List Table-->
<TR><TD ALIGN=CENTER BGCOLOR="#800000">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>Product
Type</FONT></TD>
<TD ALIGN=CENTER BGCOLOR="#800000">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>Product
Name</FONT></TD>
<TD ALIGN=CENTER WIDTH=350 BGCOLOR="#800000">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>Product
Description</FONT></TD>
<TD ALIGN=CENTER BGCOLOR="#800000">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>Unit
Price</FONT></TD></TR>
<!-- Display ADO Data Product List-->
<% Do While Not RsProductList.EOF %>
<TR> <TD BGCOLOR="f7efde" ALIGN=CENTER>
<FONT STYLE="ARIAL NARROW" SIZE=1>
```

```
<%= RsProductList("ProductType")%>
</FONT></TD>
<TD BGCOLOR="f7efde" ALIGN=CENTER>
<FONT STYLE="ARIAL NARROW" SIZE=1>
<%= RsProductList("ProductName")%>
</FONT></TD>
<TD BGCOLOR="f7efde" ALIGN=CENTER>
<FONT STYLE="ARIAL NARROW" SIZE=1>
<%= RsProductList("ProductDescription")%>
</FONT></TD>
<TD BGCOLOR="f7efde" ALIGN=CENTER>
<FONT STYLE="ARIAL NARROW" SIZE=1>
<%= RsProductList("UnitPrice")%>
</FONT></TD>
<!-- Next Row = Record -->
<%
RsProductList.MoveNext
Loop
'Remove Objects from Memory Freeing
Set RsProductList = Nothing
Set OBJdbConnection = Nothing
%>
</TABLE></FONT></Center></BODY></HTML>
```

ADO Recordset Object Delete Method

Deletes the current record or a group of records.

Delete Method Syntax

```
recordset.Delete AffectRecords
```

Delete Method Parameters

AffectRecords

An optional **AffectEnum** value that determines how many records the **Delete** method will affect. Can be one of the following constants:

Constant	Description
adAffectCurrent	Default. Delete only the current record.
adAffectGroup	Delete the records that satisfy the current ADO Recordset Object Filter Property setting. You must set the **Filter** property to one of the valid predefined constants in order to use this option. *The **Filter** property is not currently supported on UNIX.*

Delete Method Remarks

Using the **Delete** method marks the current record or a group of records in a **Recordset** object for deletion. If the **Recordset** object doesn't allow record deletion, an error occurs. If you are in immediate update mode, deletions occur in the database immediately. Otherwise, the records are marked for deletion from the cache and the actual deletion happens when you call the ADO Recordset Object UpdateBatch Method. (Use the **Filter** property to view the deleted records.)

Retrieving field values from the deleted record generates an error. After deleting the current record, the deleted record remains current until you move to a different record. Once you move away from the deleted record, it is no longer accessible.

If you nest deletions in a transaction, you can recover deleted records with the **RollbackTrans** method. If you are in batch update mode, you can cancel a pending deletion or group of pending deletions with the ADO Recordset Object CancelBatch Method.

If the attempt to delete records fails because of a conflict with the underlying data (for example, a record has already been deleted by another user), the provider returns warnings to the ADO Errors Collection, but does not halt program execution. A run-time error occurs only if there are conflicts on all the requested records.

Delete Method Examples

This VBScript example uses the **Delete** method to remove a specified record from a recordset.

```
<!-- #Include file="ADOVBS.INC" -->

<% Language = VBScript %>

<HTML>

<HEAD><TITLE>ADO 1.5 Delete Method</TITLE>

</HEAD><BODY>

<FONT FACE="MS SANS SERIF" SIZE=2>

<Center><H3>ADO Delete Method</H3>

<!--- ADO Connection Object used to create recordset-->

<%

'Create and Open Connection Object

Set OBJdbConnection = Server.CreateObject("ADODB.Connection")

OBJdbConnection.Open "AdvWorks"
```

```
'Create and Open Recordset Object
Set RsCustomerList = Server.CreateObject("ADODB.Recordset")
RsCustomerList.ActiveConnection = OBJdbConnection
RsCustomerList.CursorType = adOpenKeyset
RsCustomerList.LockType = adLockOptimistic
RsCustomerList.Source = "Customers"
RsCustomerList.Open
%>
<!-- Move to designated Record and Delete It -->
<%
If Not IsEmpty(Request.Form("WhichRecord")) Then
`Get value to move from Form Post method
Moves = Request.Form("WhichRecord")
RsCustomerList.Move CInt(Moves)
If Not RsCustomerList.EOF or RsCustomerList.BOF Then
RsCustomerList.Delete 1
RsCustomerList.MoveFirst
Else
Response.Write "Not a Valid Record Number"
RsCustomerList.MoveFirst
End If
End If
%>
<!-- BEGIN column header row for Customer Table-->
<TABLE COLSPAN=8 CELLPADDING=5 BORDER=0><TR>
<TD ALIGN=CENTER BGCOLOR="#008080">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>Company
Name</FONT>
</TD>
<TD ALIGN=CENTER BGCOLOR="#008080">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>Contact
Name</FONT>
</TD>
```

```
<TD ALIGN=CENTER WIDTH=150 BGCOLOR="#008080">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>Phone
Number</FONT>
</TD>
<TD ALIGN=CENTER BGCOLOR="#008080">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>City</FONT>
</TD>
<TD ALIGN=CENTER BGCOLOR="#008080">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff"
SIZE=1>State/Province</FONT>
</TD></TR>
<!--Display ADO Data from Customer Table Loop through Recordset
adding one Row to HTML Table each pass-->
<% Do While Not RsCustomerList.EOF %>
<TR><TD BGCOLOR="f7efde" ALIGN=CENTER>
<FONT STYLE="ARIAL NARROW" SIZE=1>
<%= RSCustomerList("CompanyName")%>
</FONT></TD>
<TD BGCOLOR="f7efde" ALIGN=CENTER>
<FONT STYLE="ARIAL NARROW" SIZE=1>
<%= RScustomerList("ContactLastName") & ", " %>
<%= RScustomerList("ContactFirstName") %>
</FONT></TD>
<TD BGCOLOR="f7efde" ALIGN=CENTER>
<FONT STYLE="ARIAL NARROW" SIZE=1>
<%= RScustomerList("PhoneNumber")%>
</FONT></TD>
<TD BGCOLOR="f7efde" ALIGN=CENTER>
<FONT STYLE="ARIAL NARROW" SIZE=1>
<%= RScustomerList("City")%>
</FONT></TD>
<TD BGCOLOR="f7efde" ALIGN=CENTER>
<FONT STYLE="ARIAL NARROW" SIZE=1>
```

```
<%= RScustomerList("StateOrProvince")%>
</FONT></TD>
</TR>
<!-Next Row = Record Loop and add to html table-->
<%
RScustomerList.MoveNext
Loop
%>
</Table></Center></FONT>
<!-- Do Client side Input Data Validation Move to named
 record and Delete it -->
<Center>
<H4>Clicking Button Will Remove Designated Record</H4>
<H5>There are <%=RsCustomerList.RecordCount - 1%> Records in this
Set</H5>
<Form Method = Post Action = "Delete.asp" Name = Form>
<Input Type = Text Name = "WhichRecord" Size = 3></Form>
<Input Type = Button Name = cmdDelete Value = "Delete
Record"></Center>
</BODY>
<Script Language = "VBScript">
Sub cmdDelete_OnClick
If IsNumeric(Document.Form.WhichRecord.Value) Then
Document.Form.WhichRecord.Value =
CInt(Document.Form.WhichRecord.Value)
Dim Response
Response = MsgBox("Are You Sure About Deleting This Record?",
vbYesNo, "ADO-ASP Example")
If Response = vbYes Then
Document.Form.Submit
End If
Else
MsgBox "You Must Enter a Valid Record Number",,"ADO-ASP Example"
```

```
End If

End Sub

</Script>

</HTML>
```

ADO Recordset Object GetRows Method

Retrieves multiple records of a recordset into an array.

GetRows Method Syntax

```
array = recordset.GetRows( Rows, Start, Fields )
```

GetRows Method Parameters

array

Two-dimensional **Array** containing records.

Rows

An optional **Long** expression indicating the number of records to retrieve. Default is **adGetRowsRest** (-1).

Start

An optional **String** or **Variant** that evaluates to the bookmark for the record from which the **GetRows** operation should begin. You can also use one of the following **BookmarkEnum** values:

Constant	Description
AdBookmarkCurrent	Start at the current record.
AdBookmarkFirst	Start at the first record.
AdBookmarkLast	Start at the last record.

Fields

An optional **Variant** representing a single field name or ordinal position or an array of field names or ordinal position numbers. ADO returns only the data in these fields.

GetRows Method Return Values

Returns a two-dimensional array.

GetRows Method Remarks

Use the **GetRows** method to copy records from a recordset into a two-dimensional array. The first subscript identifies the field and the second identifies the record number. The *array* variable is automatically dimensioned to the correct size when the **GetRows** method returns the data.

If you do not specify a value for the *Rows* argument, the **GetRows** method automatically retrieves all the records in the **Recordset** object. If you request more records than are available, **GetRows** returns only the number of available records.

If the **Recordset** object supports bookmarks, you can specify at which record the **GetRows** method should begin retrieving data by passing the value of that record's ADO Recordset Object Bookmark Property.

If you want to restrict the fields the **GetRows** call returns, you can pass either a single field name/number or an array of field names/numbers in the *Fields* argument.

After you call **GetRows**, the next unread record becomes the current record, or the ADO Recordset Object BOF, EOF Properties property is set to **True** if there are no more records.

GetRows Method Examples

This Visual Basic example uses the **GetRows** method to retrieve a specified number of rows from a recordset and to fill an array with the resulting data. The **GetRows** method will return fewer than the desired number of rows in two cases: either if **EOF** has been reached, or if **GetRows** tried to retrieve a record that was deleted by another user. The function returns **False** only if the second case occurs. The **GetRowsOK** function is required for this procedure to run.

```
Public Sub GetRowsX()

Dim rstEmployees As ADODB.Recordset

Dim strCnn As String

Dim strMessage As String

Dim intRows As Integer

Dim avarRecords As Variant

Dim intRecord As Integer

' Open recordset with names and hire dates from employee table.

strCnn = "driver={SQL Server};server=srv;" & _

"uid=sa;pwd=;database=pubs"

Set rstEmployees = New ADODB.Recordset

rstEmployees.Open "SELECT fName, lName, hire_date " & _

"FROM Employee ORDER BY lName", strCnn, , , adCmdText

Do While True

` Get user input for number of rows.

strMessage = "Enter number of rows to retrieve."

intRows = Val(InputBox(strMessage))

If intRows <= 0 Then Exit Do

` If GetRowsOK is successful, print the results,
```

```
` noting if the end of the file was reached.
If GetRowsOK(rstEmployees, intRows, _
avarRecords) Then
If intRows > UBound(avarRecords, 2) + 1 Then
Debug.Print "(Not enough records in " & _
"Recordset to retrieve " & intRows & _
" rows.)"
End If
Debug.Print UBound(avarRecords, 2) + 1 & _
" records found."
` Print the retrieved data.
For intRecord = 0 To UBound(avarRecords, 2)
Debug.Print " " & _
avarRecords(0, intRecord) & " " & _
avarRecords(1, intRecord) & ", " & _
avarRecords(2, intRecord)
Next intRecord
Else
` Assuming the GetRows error was due to data
` changes by another user, use Requery to
` refresh the Recordset and start over.
If MsgBox("GetRows failed--retry?", _
vbYesNo) = vbYes Then
rstEmployees.Requery
Else
Debug.Print "GetRows failed!"
Exit Do
End If
End If
` Because using GetRows leaves the current
` record pointer at the last record accessed,
` move the pointer back to the beginning of the
```

```
` Recordset before looping back for another search.
rstEmployees.MoveFirst
Loop
rstEmployees.Close
End Sub
Public Function GetRowsOK(rstTemp As ADODB.Recordset, _
intNumber As Integer, avarData As Variant) As Boolean
` Store results of GetRows method in array.
avarData = rstTemp.GetRows(intNumber)
` Return False only if fewer than the desired
` number of rows were returned, but not because the
` end of the Recordset was reached.
If intNumber > UBound(avarData, 2) + 1 And _
Not rstTemp.EOF Then
GetRowsOK = False
Else
GetRowsOK = True
End If
End Function
```

ADO Recordset Object Move Method

Moves the position of the current record in a **Recordset** object.

Move Method Syntax

recordset.**Move** *NumRecords*, *Start*

Move Method Parameters

NumRecords

A signed **Long** expression specifying the number of records the current record position moves.

Start

An optional **String** or **Variant** that evaluates to a bookmark. You can also use one of the following **BookmarkEnum** values:

Constant	Description
AdBookmarkCurrent	Default. Start at the current record.

AdBookmarkFirst	Start at the first record.
AdBookmarkLast	Start at the last record.

Move Method Remarks

The **Move** method is supported on all **Recordset** objects.

If the *NumRecords* argument is greater than zero, the current record position moves forward (toward the end of the recordset). If *NumRecords* is less than zero, the current record position moves backward (toward the beginning of the recordset).

If the **Move** call would move the current record position to a point before the first record, ADO sets the current record to the position before the first record in the recordset (**BOF** is **True**). An attempt to move backward when the ADO Recordset Object BOF, EOF Properties property is already **True** generates an error.

If the **Move** call would move the current record position to a point after the last record, ADO sets the current record to the position after the last record in the recordset (**EOF** is **True**). An attempt to move forward when the ADO Recordset Object BOF, EOF Properties property is already **True** generates an error.

Calling the **Move** method from an empty **Recordset** object generates an error.

If you pass the *Start* argument, the move is relative to the record with this bookmark, assuming the **Recordset** object supports bookmarks. If not specified, the move is relative to the current record.

If you are using the ADO Recordset Object CacheSize Property to locally cache records from the provider, passing a *NumRecords* that moves the current record position outside of the current group of cached records forces ADO to retrieve a new group of records starting from the destination record. The **CacheSize** property determines the size of the newly retrieved group, and the destination record is the first record retrieved.

If the **Recordset** object is forward-only, a user can still pass a *NumRecords* less than zero as long as the destination is within the current set of cached records. If the **Move** call would move the current record position to a record before the first cached record, an error will occur. Thus, you can use a record cache that supports full scrolling over a provider that only supports forward scrolling. Because cached records are loaded into memory, you should avoid caching more records than is necessary. Even if a forward-only **Recordset** object supports backward moves in this way, calling the ADO Recordset Object MoveFirst, MoveLast, MoveNext, MovePrevious Methods method on any forward-only **Recordset** object still generates an error.

Move Method Example

This VBScript example uses the **Move** method to position the record pointer based on user input. Try entering a letter or non-integer to see the error-handling work.

```
<!-- #Include file="ADOVBS.INC" -->

<% Language = VBScript %>

<HTML><HEAD>

<TITLE>ADO 1.5 Move Methods</TITLE></HEAD>
```

```
<BODY>
<FONT FACE="MS SANS SERIF" SIZE=2>
<Center>
<H3>ADO Move Methods</H3>
<%
'Create and Open Connection Object
Set OBJdbConnection = Server.CreateObject("ADODB.Connection")
OBJdbConnection.Open "AdvWorks"
'Create and Open Recordset Object
Set RsCustomerList = Server.CreateObject("ADODB.Recordset")
RsCustomerList.ActiveConnection = OBJdbConnection
RsCustomerList.CursorType = adOpenKeyset
RsCustomerList.LockType = adLockOptimistic
RsCustomerList.Source = "Customers"
RsCustomerList.Open
'Check number of user moves this session
'Increment by amount in Form
Session("Clicks") = Session("Clicks") + Request.Form("MoveAmount")
Clicks = Session("Clicks")
'Move to last known recordset position plus amount passed by Form
Post method
RsCustomerList.Move CInt(Clicks)
'Error Handling
If RsCustomerList.EOF Then
Session("Clicks") = RsCustomerList.RecordCount
Response.Write "This is the Last Record"
RsCustomerList.MoveLast
Else If RsCustomerList.BOF Then
Session("Clicks") = 1
RsCustomerList.MoveFirst
Response.Write "This is the First Record"
End If
End If
```

```
%>
<H3>Current Record Number is <BR>
<% If Session("Clicks") = 0 Then
Session("Clicks") = 1
End If
Response.Write(Session("Clicks") )%> of
<%=RsCustomerList.RecordCount%></H3>
<HR>
<Center><TABLE COLSPAN=8 CELLPADDING=5 BORDER=0>
<!-- BEGIN column header row for Customer Table-->
<TR>
<TD ALIGN=CENTER BGCOLOR="#008080">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>Company
Name</FONT>
</TD>
<TD ALIGN=CENTER BGCOLOR="#008080">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>Contact
Name</FONT>
</TD>
<TD ALIGN=CENTER WIDTH=150 BGCOLOR="#008080">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>Phone
Number</FONT>
</TD>
<TD ALIGN=CENTER BGCOLOR="#008080">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>City</FONT>
</TD>
<TD ALIGN=CENTER BGCOLOR="#008080">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff"
SIZE=1>State/Province</FONT>
</TD>
</TR>
<!--Display ADO Data from Customer Table-->
<TR>
<TD BGCOLOR="f7efde" ALIGN=CENTER>
```

```
<FONT STYLE="ARIAL NARROW" SIZE=1>
<%= RSCustomerList("CompanyName")%>
</FONT></TD>
<TD BGCOLOR="f7efde" ALIGN=CENTER>
<FONT STYLE="ARIAL NARROW" SIZE=1>
<%= RScustomerList("ContactLastName") & ", " %>
<%= RScustomerList("ContactFirstName") %>
</FONT></TD>
<TD BGCOLOR="f7efde" ALIGN=CENTER>
<FONT STYLE="ARIAL NARROW" SIZE=1>
<%= RScustomerList("PhoneNumber")%>
</FONT></TD>
<TD BGCOLOR="f7efde" ALIGN=CENTER>
<FONT STYLE="ARIAL NARROW" SIZE=1>
<%= RScustomerList("City")%>
</FONT></TD>
<TD BGCOLOR="f7efde" ALIGN=CENTER>
<FONT STYLE="ARIAL NARROW" SIZE=1>
<%= RScustomerList("StateOrProvince")%>
</FONT></TD>
</TR> </Table></FONT>
<HR>
<Input Type = Button Name = cmdDown Value = "< ">
<Input Type = Button Name = cmdUp Value = " >">
<H5>Click Direction Arrows for Previous or Next Record
<BR> Click Move Amount to use Move Method
Enter Number of Records to Move + or - </H5>
<Table>
<Form Method = Post Action="Move.asp" Name=Form>
<TR><TD><Input Type="Button" Name = Move Value="Move Amount
"></TD><TD></TD><TD>
<Input Type="Text" Size="4" Name="MoveAmount" Value = 0></TD><TR>
</Form></Table></Center>
```

```
</BODY>
<Script Language = "VBScript">
Sub Move_OnClick
' Make sure move value entered is an integer
If IsNumeric(Document.Form.MoveAmount.Value)Then
Document.Form.MoveAmount.Value =
CInt(Document.Form.MoveAmount.Value)
Document.Form.Submit
Else
MsgBox "You Must Enter a Number", ,"ADO-ASP Example"
Document.Form.MoveAmount.Value = 0
End If
End Sub
Sub cmdDown_OnClick
Document.Form.MoveAmount.Value = -1
Document.Form.Submit
End Sub
Sub cmdUp_OnClick
Document.Form.MoveAmount.Value = 1
Document.Form.Submit
End Sub
</Script>
</HTML>
```

ADO Recordset Object MoveFirst, MoveLast, MoveNext, MovePrevious Methods

These methods move to the first, last, next, or previous record in a specified **Recordset** object and make that record the current record.

MoveFirst, MoveLast, MoveNext, MovePrevious Methods Syntax

recordset.**{MoveFirst | MoveLast | MoveNext | MovePrevious}**

MoveFirst, MoveLast, MoveNext, MovePrevious Methods Remarks

Use the **MoveFirst** method to move the current record position to the first record in the recordset.

Use the **MoveLast** method to move the current record position to the last record in the recordset. The **Recordset** object must support bookmarks or backward cursor movement; otherwise, the method call will generate an error.

Use the **MoveNext** method to move the current record position one record forward (toward the bottom of the recordset). If the last record is the current record and you call the **MoveNext** method, ADO sets the current record to the position after the last record in the recordset (**EOF** is **True**). An attempt to move forward when the ADO Recordset Object BOF, EOF Properties property is already **True** generates an error.

Use the **MovePrevious** method to move the current record position one record backward (toward the top of the recordset). The **Recordset** object must support bookmarks or backward cursor movement; otherwise, the method call will generate an error. If the first record is the current record and you call the **MovePrevious** method, ADO sets the current record to the position before the first record in the recordset (**BOF** is **True**). An attempt to move backward when the ADO Recordset Object BOF, EOF Properties property is already **True** generates an error. If the **Recordset** object does not support either bookmarks or backward cursor movement, the **MovePrevious** method will generate an error.

If the recordset is forward-only and you want to support both forward and backward scrolling, you can use the ADO Recordset Object CacheSize Property to create a record cache that will support backward cursor movement through the ADO Recordset Object Move Method. Because cached records are loaded into memory, you should avoid caching more records than is necessary. You can call the **MoveFirst** method in a forward-only **Recordset** object; doing so may cause the provider to re-execute the command that generated the **Recordset** object.

MoveFirst, MoveLast, MoveNext, MovePrevious Methods Example

This VBScript example uses the **MoveFirst**, **MoveLast**, **MoveNext**, and **MovePrevious** methods to move the record pointer of a recordset based on the supplied command. The **MoveAny** function is required for this procedure to run. Try moving beyond the upper or lower limits of the recordset to see error-handling work.

```
<!-- #Include file="ADOVBS.INC" -->

<% Language = VBScript %>

<HTML><HEAD>

<TITLE>ADO 1.5 MoveNext MovePrevious MoveLast MoveFirst
Methods</TITLE></HEAD>

<BODY>

<FONT FACE="MS SANS SERIF" SIZE=2>

<Center>

<H3>ADO Methods<BR>MoveNext MovePrevious MoveLast MoveFirst</H3>

<!-- Create Connection and Recordset Objects on Server -->

<%

'Create and Open Connection Object
```

```
Set OBJdbConnection = Server.CreateObject("ADODB.Connection")
OBJdbConnection.Open "AdvWorks"
'Create and Open Recordset Object
Set RsCustomerList = Server.CreateObject("ADODB.Recordset")
RsCustomerList.ActiveConnection = OBJdbConnection
RsCustomerList.CursorType = adOpenKeyset
RsCustomerList.LockType = adLockOptimistic
RsCustomerList.Source = "Customers"
RsCustomerList.Open
' Check Request.Form collection to see if any moves are recorded
If Not IsEmpty(Request.Form("MoveAmount")) Then
'Keep track of the number and direction of moves this session
Session("Moves") = Session("Moves") + Request.Form("MoveAmount")
Clicks = Session("Moves")
'Move to last known position
RsCustomerList.Move CInt(Clicks)
'Check if move is + or - and do error checking
If CInt(Request.Form("MoveAmount")) = 1 Then
If RsCustomerList.EOF Then
Session("Moves") = RsCustomerList.RecordCount
RsCustomerList.MoveLast
End If
RsCustomerList.MoveNext
End If
If Request.Form("MoveAmount") < 1 Then
RsCustomerList.MovePrevious
End If
'Check if First Record or Last Record Command Buttons Clicked
If Request.Form("MoveLast") = 3 Then
RsCustomerList.MoveLast
Session("Moves") = RsCustomerList.RecordCount
End If
```

```
If Request.Form("MoveFirst") = 2 Then
RsCustomerList.MoveFirst
Session("Moves") = 1
End If
End If
' Do Error checking for combination of Move Button clicks
If RsCustomerList.EOF Then
Session("Moves") = RsCustomerList.RecordCount
RsCustomerList.MoveLast
Response.Write "This is the Last Record"
End If
If RsCustomerList.BOF Then
Session("Moves") = 1
RsCustomerList.MoveFirst
Response.Write "This is the First Record"
End If
%>
<H3>Current Record Number is <BR>
<!-- Display Current Record Number and Recordset Size -->
<% If IsEmpty(Session("Moves")) Then
Session("Moves") = 1
End If
%>
<%Response.Write(Session("Moves") )%> of
<%=RsCustomerList.RecordCount%></H3>
<HR>
<Center><TABLE COLSPAN=8 CELLPADDING=5 BORDER=0>
<!-- BEGIN column header row for Customer Table-->
<TR><TD ALIGN=CENTER BGCOLOR="#008080">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>Company
Name</FONT>
</TD>
<TD ALIGN=CENTER BGCOLOR="#008080">
```

```
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>Contact
Name</FONT>

</TD>

<TD ALIGN=CENTER WIDTH=150 BGCOLOR="#008080">

<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>Phone
Number</FONT>

</TD>

<TD ALIGN=CENTER BGCOLOR="#008080">

<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=1>City</FONT>

</TD>

<TD ALIGN=CENTER BGCOLOR="#008080">

<FONT STYLE="ARIAL NARROW" COLOR="#ffffff"
SIZE=1>State/Province</FONT>

</TD></TR>

<!--Display ADO Data from Customer Table-->

<TR>

<TD BGCOLOR="f7efde" ALIGN=CENTER>

<FONT STYLE="ARIAL NARROW" SIZE=1>

<%= RSCustomerList("CompanyName")%>

</FONT></TD>

<TD BGCOLOR="f7efde" ALIGN=CENTER>

<FONT STYLE="ARIAL NARROW" SIZE=1>

<%= RScustomerList("ContactLastName") & ", " %>

<%= RScustomerList("ContactFirstName") %>

</FONT></TD>

<TD BGCOLOR="f7efde" ALIGN=CENTER>

<FONT STYLE="ARIAL NARROW" SIZE=1>

<%= RScustomerList("PhoneNumber")%>

</FONT></TD>

<TD BGCOLOR="f7efde" ALIGN=CENTER>

<FONT STYLE="ARIAL NARROW" SIZE=1>

<%= RScustomerList("City")%>

</FONT></TD>
```

```
<TD BGCOLOR="f7efde" ALIGN=CENTER>
<FONT STYLE="ARIAL NARROW" SIZE=1>
<%= RScustomerList("StateOrProvince")%>
</FONT></TD>
</TR> </Table></FONT>
<HR>
<Input Type = Button Name = cmdDown Value = "< ">
<Input Type = Button Name = cmdUp Value = " >">
<BR>
<Input Type = Button Name = cmdFirst Value = "First Record">
<Input Type = Button Name = cmdLast Value = "Last Record">
<H5>Click Direction Arrows to Use MovePrevious or MoveNext
<BR> </H5>
<!-- Use Hidden Form Fields to send values to Server -->
<Form Method = Post Action="MoveOne.asp" Name=Form>
<Input Type="Hidden" Size="4" Name="MoveAmount" Value = 0>
<Input Type="Hidden" Size="4" Name="MoveLast" Value = 0>
<Input Type="Hidden" Size="4" Name="MoveFirst" Value = 0>
</Form></BODY>
<Script Language = "VBScript">
Sub cmdDown_OnClick
'Set Values in Form Input Boxes and Submit Form
Document.Form.MoveAmount.Value = -1
Document.Form.Submit
End Sub
Sub cmdUp_OnClick
Document.Form.MoveAmount.Value = 1
Document.Form.Submit
End Sub
Sub cmdFirst_OnClick
Document.Form.MoveFirst.Value = 2
Document.Form.Submit
```

```
End Sub

Sub cmdLast_OnClick

Document.Form.MoveLast.Value = 3

Document.Form.Submit

End Sub

</Script></HTML>
```

ADO Recordset Object NextRecordset Method

Clears the current **Recordset** object and returns the next recordset by advancing through a series of commands. *This method is not currently supported on UNIX.*

NextRecordset Method Syntax

```
Set recordset2 = recordset1.NextRecordset( RecordsAffected )
```

NextRecordset Method Parameters

recordset2

Recordset containing results of command.

RecordsAffected

An optional **Long** variable to which the provider returns the number of records that the current operation affected.

NextRecordset Method Return Values

Returns a **Recordset** object. In the syntax model, *recordset1* and *recordset2* can be the same **Recordset** object, or you can use separate objects.

NextRecordset Method Remarks

Use the **NextRecordset** method to return the results of the next command in a compound command statement or of a stored procedure that returns multiple results. If you open a **Recordset** object based on a compound command statement (for example, `"SELECT * FROM table1;SELECT * FROM table2"`) using the ADO Command Object Execute Method on an ADO Command Object or the ADO Recordset Object Open Method on a recordset, ADO executes only the first command and returns the results to *recordset*. To access the results of subsequent commands in the statement, call the **NextRecordset** method.

As long as there are additional results, the **NextRecordset** method will continue to return **Recordset** objects. If a row-returning command returns no records, the returned **Recordset** object will be empty; test for this case by verifying that the ADO Recordset Object BOF, EOF Properties are both **True**. If a non row-returning command executes successfully, the returned **Recordset** object will be closed, which you can verify by testing the ADO Recordset Object State Property on the recordset. When there are no more results, *recordset* will be set to *Nothing*.

If an edit is in progress while in immediate update mode, calling the **NextRecordset** method generates an error; call the ADO Recordset Object Update Method or the ADO Recordset Object CancelUpdate Method first.

If you need to pass parameters for more than one command in the compound statement by filling the ADO Parameters Collection or by passing an array with the original **Open** or **Execute** call, the parameters must be in the same order in the collection or array as their respective commands in the command series. You must finish reading all the results before reading output parameter values.

When you call the **NextRecordset** method, ADO executes only the next command in the statement. If you explicitly close the **Recordset** object before stepping through the entire command statement, ADO never executes the remaining commands.

The **NextRecordset** method is not available on a client-side (ADOR) **Recordset** object.

NextRecordset Method Example

This Visual Basic example uses the **NextRecordset** method to view the data in a recordset that uses a compound command statement made up of three separate **SELECT** statements.

```
Public Sub NextRecordsetX()

Dim rstCompound As ADODB.Recordset

Dim strCnn As String

Dim intCount As Integer

` Open compound recordset.

strCnn = "driver={SQL Server};server=srv;" & _

"uid=sa;pwd=;database=pubs"

Set rstCompound = New ADODB.Recordset

rstCompound.Open "SELECT * FROM authors; " & _

"SELECT * FROM stores; " & _

"SELECT * FROM jobs", strCnn, , , adCmdText

` Display results from each SELECT statement.

intCount = 1

Do Until rstCompound Is Nothing

Debug.Print "Contents of recordset #" & intCount

Do While Not rstCompound.EOF

Debug.Print , rstCompound.Fields(0), _

rstCompound.Fields(1)

rstCompound.MoveNext

Loop
```

```
Set rstCompound = rstCompound.NextRecordset
intCount = intCount + 1
Loop
End Sub
```

ADO Recordset Object Open Method

Opens a cursor.

Open Method Syntax

```
recordset.Open Source, ActiveConnection, CursorType, LockType,
Options
```

Open Method Parameters

Source

An optional **Variant** that evaluates to a valid **Command** object variable name, an SQL statement, a table name, or a stored procedure call.

ActiveConnection

An optional **Variant** that evaluates to a valid **Connection** object variable name, or a **String** containing **ConnectionString** parameters.

CursorType

An optional **CursorTypeEnum** value that determines the type of cursor that the provider should use when opening the recordset. Can be one of the following constants (See the ADO Recordset Object CursorType Property for definitions of these settings.):

Constant	Description
adOpenForwardOnly	Default. Opens a forward-only cursor.
adOpenKeyset	Opens a keyset cursor.
adOpenDynamic	Opens a dynamic cursor.
adOpenStatic	Opens a static cursor.

LockType

An optional **LockTypeEnum** value that determines what type of locking (concurrency) the provider should use when opening the recordset. Can be one of the following constants (See the **LockType** property for more information.):

Constant	Description
adLockReadOnly	Default. Read-only; you cannot alter the data.
adLocPessimistic	Pessimistic locking, record by record. The provider does what is necessary to ensure successful editing of the records, usually by

	locking records at the data source immediately upon editing.
adLockOptimistic	Optimistic locking, record by record. The provider uses optimistic locking, locking records only when you call the **Update** method.
adLockBatchOptimistic	Optimistic batch updates. Required for batch update mode as opposed to immediate update mode.

Options

An optional **Long** value that indicates how the provider should evaluate the *Source* argument if it represents something other than a **Command** object. Can be one of the following constants (See the **CommandType** property for a more detailed explanation of these constants.):

Constant	Description
adCmdText	The provider should evaluate *Source* as a textual definition of a command.
adCmdTable	The provider should evaluate *Source* as a table name.
adCmdStoredProc	The provider should evaluate *Source* as a stored procedure.
adCmdUnknown	The type of command in the *Source* argument is not known.

See the ADO Command Object CommandType Property for a more detailed explanation of the four constants in this list.

Open Method Remarks

Using the **Open** method on a **Recordset** object opens a cursor that represents records from a base table or the results of a query.

Use the optional *Source* argument to specify a data source using one of the following: an ADO Command Object variable, an SQL statement, a stored procedure, or a table name.

The *ActiveConnection* argument corresponds to the **ActiveConnection** property and specifies in which connection to open the **Recordset** object. If you pass a connection definition for this argument, ADO opens a new connection using the specified parameters. You can change the value of this property after opening the recordset to send updates to another provider. Or, you can set this property to **Nothing** (in Microsoft Visual Basic) to disconnect the recordset from any provider.

For the other arguments that correspond directly to properties of a **Recordset** object (*Source*, *CursorType*, and *LockType*), the relationship of the arguments to the properties is as follows:

- The property is read/write before the **Recordset** object is opened.
- The property settings are used unless you pass the corresponding arguments when executing the **Open** method. If you pass an argument, it overrides the corresponding property setting, and the property setting is updated with the argument value.
- After you open the **Recordset** object, these properties become read-only.

Note

> For **Recordset** objects whose ADO Recordset Object Source Property is set to a valid **Command** object, the **ActiveConnection** property is read-only, even if the **Recordset** object isn't open.

If you pass a **Command** object in the *Source* argument and also pass an *ActiveConnection* argument, an error occurs. The **ActiveConnection** property of the **Command** object must already be set to a valid ADO Connection Object or connection string.

If you pass something other than a **Command** object in the *Source* argument, you can use the *Options* argument to optimize evaluation of the *Source* argument. If the *Options* argument is not defined, you may experience diminished performance because ADO must make calls to the provider to determine if the argument is an SQL statement, a stored procedure, or a table name. If you know what *Source* type you're using, setting the *Options* argument instructs ADO to jump directly to the relevant code. If the *Options* argument does not match the *Source* type, an error occurs.

If the data source returns no records, the provider sets both the ADO Recordset Object BOF, EOF Properties to **True**, and the current record position is undefined. You can still add new data to this empty **Recordset** object if the cursor type allows it.

When you have concluded your operations over an open **Recordset** object, use the ADO Recordset Object Close Method to free any associated system resources. Closing an object does not remove it from memory; you may change its property settings and use the **Open** method to open it again later. To completely eliminate an object from memory, set the object variable to *Nothing*.

Open Method Examples

See the ADO Recordset Object Close Method.

ADO Recordset Object Requery Method

Updates the data in a **Recordset** object by re-executing the query on which the object is based.

Requery Method Syntax

```
recordset.Requery
```

Requery Method Remarks

Use the **Requery** method to refresh the entire contents of a **Recordset** object from the data source by reissuing the original command and retrieving the data a second time. Calling this method is equivalent to calling the ADO Recordset Object Close Method and ADO Recordset Object Open Method methods in succession. If you are editing the current record or adding a new record, an error occurs.

While the **Recordset** object is open, the properties that define the nature of the cursor (ADO Recordset Object CursorType Property**,** ADO Recordset Object LockType Property, ADO Recordset Object MaxRecords Property, and so forth) are read-only. Thus, the **Requery** method can only refresh the current cursor. To change any of the cursor properties and view the results,

you must use the **Close** method so that the properties become read/write again. You can then change the property settings and call the **Open** method to reopen the cursor.

Requery Method Example

See the command ADO Command Object Execute Method.

ADO Recordset Object Resync Method

Refreshes the data in the current **Recordset** object from the underlying database.

Resync Method Syntax

```
recordset.Resync AffectRecords
```

Resync Method Parameters

AffectRecords

An optional **AffectEnum** constant that determines how many records the **Resync** method will affect. Can be one of the following constants:

Constant	Description
adAffectCurrent	Refresh only the current record.
adAffectGroup	Refresh the records that satisfy the current **Filter** property setting. You must set the **Filter** property to one of the valid predefined constants in order to use this option. *The **Filter** property is not currently supported on UNIX.*
adAffectAll	Default. Refresh all the records in the **Recordset** object, including any hidden by the current **Filter** property setting.

Resync Method Remarks

Use the **Resync** method to re-synchronize records in the current recordset with the underlying database. This is useful if you are using either a static or forward-only cursor but you want to see any changes in the underlying database. Calling the **Resync** method cancels any pending batch updates.

Unlike the ADO Recordset Object Requery Method, the **Resync** method does not re-execute the **Recordset** object's underlying command; new records in the underlying database will not be visible.

If the attempt to resynchronize fails because of a conflict with the underlying data (for example, a record has been deleted by another user), the provider returns warnings to the ADO Errors Collection, but does not halt program execution. A run-time error occurs only if there are conflicts on all the requested records. Use the ADO Recordset Object Filter Property (**adFilterAffectedRecords**) and the ADO Recordset Object Status Property to locate records with conflicts.

Resync Method Examples

This Visual Basic example demonstrates using the **Resync** method to refresh data in a static recordset.

```
Public Sub ResyncX()
Dim strCnn As String
Dim rstTitles As ADODB.Recordset
' Open connections.
strCnn = "driver={SQL Server};server=srv;" & _
"uid=sa;pwd=;database=pubs"
' Open recordset for titles table.
Set rstTitles = New ADODB.Recordset
rstTitles.CursorType = adOpenStatic
rstTitles.LockType = adLockBatchOptimistic
rstTitles.Open "titles", strCnn, , , adCmdTable
' Change the type of the first title in the recordset.
rstTitles!Type = "database"
' Display the results of the change.
MsgBox "Before resync: " & vbCr & vbCr & _
"Title - " & rstTitles!Title & vbCr & _
"Type - " & rstTitles!Type
' Resync with database and redisplay results.
rstTitles.Resync
MsgBox "After resync: " & vbCr & vbCr & _
"Title - " & rstTitles!Title & vbCr & _
"Type - " & rstTitles!Type
rstTitles.CancelBatch
rstTitles.Close
End Sub
```

ADO Recordset Object Supports Method

Determines whether a specified **Recordset** object supports a particular type of functionality.

Supports Method Syntax

```
boolean = recordset.Supports( CursorOptions )
```

Supports Method Parameters

CursorOptions

A **Long** expression that consists of one or more of the following **CursorOptionEnum** values:

Value	Description
adAddNew	The **AddNew** method adds new records.
adApproxPosition	You can read and set the **AbsolutePosition** and **AbsolutePage** properties.
adBookmark	The **Bookmark** property accesses specific records.
adDelete	The **Delete** method deletes records.
adHoldRecords	You can retrieve more records or change the next retrieve position without committing all pending changes.
adMovePrevious	The **MoveFirst**, **MovePrevious**, **Move**, and **GetRows** methods move the current position backward without requiring bookmarks.
adResync	The **Resync** method modifies existing data.
adUpdate	The **Update** method modifies existing data.
adUpdateBatch	The **UpdateBatch** and **CancelBatch** methods transmit changes to the provider in groups.

Supports Method Remarks

Use the **Supports** method to determine what types of functionality a **Recordset** object supports. If the **Recordset** object supports the features whose corresponding constants are in *CursorOptions*, the **Supports** method returns **True**. Otherwise, it returns **False**.

Note

Although the **Supports** method may return **True** for a given functionality, it does not guarantee that the provider can make the feature available under all circumstances. The **Supports** method simply returns whether or not the provider can support the specified functionality assuming certain conditions are met. For example, the **Supports** method may indicate that a **Recordset** object supports updates even though the cursor is based on a multi-table join, some columns of which are not updatable.

Supports Method Examples

This Visual Basic example uses the **Supports** method to display the options supported by a recordset opened with different cursor types. The **DisplaySupport** function is required for this procedure to run.

```
Public Sub SupportsX()

Dim aintCursorType(4) As Integer

Dim rstTitles As ADODB.Recordset

Dim strCnn As String
```

```
Dim intIndex As Integer
` Open connections.
strCnn = "driver={SQL Server};server=srv;" & _
"uid=sa;pwd=;database=pubs"
` Fill array with CursorType constants.
aintCursorType(0) = adOpenForwardOnly
aintCursorType(1) = adOpenKeyset
aintCursorType(2) = adOpenDynamic
aintCursorType(3) = adOpenStatic
` Open recordset using each CursorType and
` optimitic locking. Then call the DisplaySupport
` procedure to display the supported options.
For intIndex = 0 To 3
Set rstTitles = New ADODB.Recordset
rstTitles.CursorType = aintCursorType(intIndex)
rstTitles.LockType = adLockOptimistic
rstTitles.Open "titles", strCnn, , , adCmdTable
Select Case aintCursorType(intIndex)
Case adOpenForwardOnly
Debug.Print "ForwardOnly cursor supports:"
Case adOpenKeyset
Debug.Print "Keyset cursor supports:"
Case adOpenDynamic
Debug.Print "Dynamic cursor supports:"
Case adOpenStatic
Debug.Print "Static cursor supports:"
End Select
DisplaySupport rstTitles
rstTitles.Close
Next intIndex
End Sub
Public Sub DisplaySupport(rstTemp As ADODB.Recordset)
```

```
Dim alngConstants(9) As Long
Dim booSupports As Boolean
Dim intIndex As Integer
' Fill array with cursor option constants.
alngConstants(0) = adAddNew
alngConstants(1) = adApproxPosition
alngConstants(2) = adBookmark
alngConstants(3) = adDelete
alngConstants(4) = adHoldRecords
alngConstants(5) = adMovePrevious
alngConstants(6) = adResync
alngConstants(7) = adUpdate
alngConstants(8) = adUpdateBatch
For intIndex = 0 To 8
booSupports = _
rstTemp.Supports(alngConstants(intIndex))
If booSupports Then
Select Case alngConstants(intIndex)
Case adAddNew
Debug.Print " AddNew"
Case adApproxPosition
Debug.Print " AbsolutePosition and AbsolutePage"
Case adBookmark
Debug.Print " Bookmark"
Case adDelete
Debug.Print " Delete"
Case adHoldRecords
Debug.Print " holding records"
Case adMovePrevious
Debug.Print " MovePrevious and Move"
Case adResync
Debug.Print " resyncing data"
```

```
Case adUpdate
Debug.Print " Update"
Case adUpdateBatch
Debug.Print " batch updating"
End Select
End If
Next intIndex
End Sub
```

ADO Recordset Object Update Method

Saves any changes you make to the current record of a **Recordset** object.

Note

This method is not available for some databases and ODBC drivers.

Update Method Syntax

```
recordset.Update Fields, Values
```

Update Method Parameters

Fields

An optional **Variant** representing a single name or a **Variant** array representing names or ordinal positions of the field or fields you wish to modify.

Values

An optional **Variant** representing a single value or a **Variant** array representing values for the field or fields in the new record.

Update Method Remarks

Use the **Update** method to save any changes you make to the current record of a **Recordset** object since calling the ADO Recordset Object AddNew Method or since changing any field values in an existing record. The **Recordset** object must support updates.

To set field values, do one of the following:

- Assign values to a ADO Field Object object's ADO Field Object Value Property and call the ADO Recordset Object Update Method.
- Pass a field name and a value as arguments with the **Update** call.
- Pass an array of field names and an array of values with the **Update** call.

When you use arrays of fields and values, there must be an equal number of elements in both arrays. Also, the order of field names must match the order of field values. If the number and order of fields and values do not match, an error occurs.

If the **Recordset** object supports batch updating, then you can cache multiple changes to one or more records locally until you call the ADO Recordset Object UpdateBatch Method. If you are editing the current record or adding a new record when you call the **UpdateBatch** method, ADO will automatically call the **Update** method to save any pending changes to the current record before transmitting the batched changes to the provider. *Batch updating is not currently supported on UNIX.*

If you move from the record you are adding or editing before calling the **Update** method, ADO will automatically call **Update** to save the changes. You must call the ADO Recordset Object CancelUpdate Method if you want to cancel any changes made to the current record or to discard a newly added record.

The current record remains current after you call the **Update** method.

Update Method Examples

The following Visual Basic examples show how to use the **Update** method.

The first example demonstrates using the **Update** method in conjunction with **CancelUpdate** method.

```
Public Sub UpdateX()
Dim rstEmployees As ADODB.Recordset
Dim strOldFirst As String
Dim strOldLast As String
Dim strMessage As String
` Open recordset with names from Employee table.
strCnn = "driver={SQL Server};server=srv;" & _
"uid=sa;pwd=;database=pubs"
Set rstEmployees = New ADODB.Recordset
rstEmployees.CursorType = adOpenKeyset
rstEmployees.LockType = adLockOptimistic
rstEmployees.Open "SELECT fname, lname " & _
"FROM Employee ORDER BY lname", strCnn, , , adCmdText
` Store original data.
strOldFirst = rstEmployees!fname
strOldLast = rstEmployees!lname
` Change data in edit buffer.
```

```
rstEmployees!fname = "Linda"
rstEmployees!lname = "Kobara"
` Show contents of buffer and get user input.
strMessage = "Edit in progress:" & vbCr & _
" Original data = " & strOldFirst & " " & _
strOldLast & vbCr & " Data in buffer = " & _
rstEmployees!fname & " " & rstEmployees!lname & vbCr & vbCr & _
"Use Update to replace the original data with " & _
"the buffered data in the Recordset?"
If MsgBox(strMessage, vbYesNo) = vbYes Then
rstEmployees.Update
Else
rstEmployees.CancelUpdate
End If
` Show the resulting data.
MsgBox "Data in recordset = " & rstEmployees!fname & " " & _
rstEmployees!lname
` Restore original data because this is a demonstration.
If Not (strOldFirst = rstEmployees!fname And _
strOldLast = rstEmployees!lname) Then
rstEmployees!fname = strOldFirst
rstEmployees!lname = strOldLast
rstEmployees.Update
End If
rstEmployees.Close
End Sub
```

The following example demonstrates using the **Update** method in conjunction with the **AddNew** method:

```
Public Sub UpdateX2()
Dim cnn1 As ADODB.Connection
Dim rstEmployees As ADODB.Recordset
Dim strEmpID As String
Dim strOldFirst As String
```

```
Dim strOldLast As String
Dim strMessage As String
' Open a connection.
Set cnn1 = New ADODB.Connection
strCnn = "driver={SQL Server};server=srv;" & _
"uid=sa;pwd=;database=pubs"
cnn1.Open strCnn
' Open recordset with data from Employee table.
Set rstEmployees = New ADODB.Recordset
rstEmployees.CursorType = adOpenKeyset
rstEmployees.LockType = adLockOptimistic
rstEmployees.Open "employee", cnn1, , , adCmdTable
rstEmployees.AddNew
strEmpID = "B-S55555M"
rstEmployees!emp_id = strEmpID
rstEmployees!fname = "Bill"
rstEmployees!lname = "Sornsin"
' Show contents of buffer and get user input.
strMessage = "AddNew in progress:" & vbCr & _
"Data in buffer = " & rstEmployees!emp_id & ", " & _
rstEmployees!fname & " " & rstEmployees!lname & vbCr & vbCr & _
"Use Update to save buffer to recordset?"
If MsgBox(strMessage, vbYesNoCancel) = vbYes Then
rstEmployees.Update
` Go to the new record and show the resulting data.
MsgBox "Data in recordset = " & rstEmployees!emp_id & ", " & _
rstEmployees!fname & " " & rstEmployees!lname
Else
rstEmployees.CancelUpdate
MsgBox "No new record added."
End If
' Delete new data because this is a demonstration.
```

```
cnn1.Execute "DELETE FROM employee WHERE emp_id = '" & strEmpID &
"'"

rstEmployees.Close

End Sub
```

ADO Recordset Object UpdateBatch Method

Writes all pending batch updates to disk. *This method is not currently supported on UNIX.*

UpdateBatch Method Syntax

```
recordset.UpdateBatch AffectRecords
```

UpdateBatch Method Parameters

AffectRecords

An optional **AffectEnum** value that determines how many records the **UpdateBatch** method will affect. Can be one of the following constants:

Constant	Description
AdAffectCurrent	Write pending changes only for the current record.
AdAffectGroup	Write pending changes for the records that satisfy the current **Filter** property setting. You must set the **Filter** property to one of the valid predefined constants in order to use this option.
adAffectAll	Default. Write pending changes for all the records in the **Recordset** object, including any hidden by the current **Filter** property setting.

UpdateBatch Method Remarks

Use the **UpdateBatch** method when modifying a **Recordset** object in batch update mode to transmit all changes made in a **Recordset** object to the underlying database.

If the **Recordset** object supports batch updating, then you can cache multiple changes to one or more records locally until you call the **UpdateBatch** method. If you are editing the current record or adding a new record when you call the **UpdateBatch** method, ADO will automatically call the ADO Recordset Object Update Method to save any pending changes to the current record before transmitting the batched changes to the provider.

Note

You should use batch updating only with either a keyset or static cursor.

If the attempt to transmit changes fails because of a conflict with the underlying data (for example, a record has already been deleted by another user), the provider returns warnings to the ADO Errors Collection but does not halt program execution. A run-time error occurs only if there are conflicts on all the requested records. Use the ADO Recordset Object Filter Property

(**adFilterAffectedRecords**) and the ADO Recordset Object Status Property to locate records with conflicts.

To cancel all pending batch updates, use the ADO Recordset Object CancelBatch Method.

UpdateBatch Method Example

See the ADO Recordset Object CancelBatch Method example.

ADO Recordset Object Properties

ADO Recordset Object AbsolutePage Property

Specifies in which page the current record resides.

AbsolutePage Property Return Values

Sets or returns a **Long** value from 1 to the number of pages in the **Recordset** object (**PageCount**), or returns one of the following constants:

Constant	Description
adPosUnknown	The recordset is empty, the current position is unknown, or the provider does not support the **AbsolutePage** property
adPosBOF	The current record pointer is at BOF (that is, the **BOF** property is **True**).
adPosEOF	The current record pointer is at EOF (that is, the **EOF** property is **True**).

AbsolutePage Property Remarks

Use the **AbsolutePage** property to identify the page number on which the current record is located. Use the ADO Recordset Object PageSize Property to logically divide the **Recordset** object into a series of pages, each of which has the number of records equal to **PageSize** (except for the last page, which may have fewer records). The provider must support the appropriate functionality for this property to be available.

Like the **AbsolutePosition** property, **AbsolutePage** is 1-based and equals 1 when the current record is the first record in the recordset. Set this property to move to the first record of a particular page. Obtain the total number of pages from the ADO Recordset Object PageCount Property.

AbsolutePage Property Example

This Visual Basic example uses the **AbsolutePage**, **PageCount**, and **PageSize** properties to display names and hire dates from the ***Employees*** table five records at a time.

```
Public Sub AbsolutePageX()

Dim rstEmployees As ADODB.Recordset

Dim strCnn As String

Dim strMessage As String
```

```
Dim intPage As Integer
Dim intPageCount As Integer
Dim intRecord As Integer
` Open a recordset using a client cursor
` for the employee table.
strCnn = "driver={SQL Server};server=srv;" & _
"uid=sa;pwd=;database=pubs"
Set rstEmployees = New ADODB.Recordset
` Use client cursor to enable AbsolutePosition property.
rstEmployees.CursorLocation = adUseClient
rstEmployees.Open "employee", strCnn, , , adCmdTable

` Display names and hire dates, five records
` at a time.
rstEmployees.PageSize = 5
intPageCount = rstEmployees.PageCount
For intPage = 1 To intPageCount
rstEmployees.AbsolutePage = intPage
strMessage = ""
For intRecord = 1 To rstEmployees.PageSize
strMessage = strMessage & _
rstEmployees!fname & " " & _
rstEmployees!lname & " " & _
rstEmployees!hire_date & vbCr
rstEmployees.MoveNext
If rstEmployees.EOF Then Exit For
Next intRecord
MsgBox strMessage
Next intPage
rstEmployees.Close
End Sub
```

ADO Recordset Object AbsolutePosition Property

Specifies the ordinal position of a **Recordset** object's current record.

AbsolutePosition Property Return Values

Sets or returns a **Long** value from 1 to the number of records in the **Recordset** object (**RecordCount**), or returns one of the following constants:

Constant	Description
adPosUnknown	The recordset is empty, the current position is unknown, or the provider does not support the **AbsolutePosition** property.
adPosBOF	The current record pointer is at BOF (that is, the **BOF** property is **True**).
adPosEOF	The current record pointer is at EOF (that is, the **EOF** property is **True**).

AbsolutePosition Property Remarks

Use the **AbsolutePosition** property to move to a record based on its ordinal position in the **Recordset** object, or to determine the ordinal position of the current record. The provider must support the appropriate functionality for this property to be available.

Like the **AbsolutePage** property, **AbsolutePosition** is 1-based and equals 1 when the current record is the first record in the recordset. You can obtain the total number of records in the **Recordset** object from the **RecordCount** property.

When you set the **AbsolutePosition** property, even if it is to a record in the current cache, ADO reloads the cache with a new group of records starting with the record you specified. The ADO Recordset Object CacheSize Property determines the size of this group.

Note

> You should not use the **AbsolutePosition** property as a surrogate record number. The position of a given record changes when you delete a preceding record. There is also no assurance that a given record will have the same **AbsolutePosition** if the **Recordset** object is requeried or reopened. Bookmarks are still the recommended way of retaining and returning to a given position, and are the only way of positioning across all types of **Recordset** objects.

AbsolutePosition Property Example

This Visual Basic example demonstrates how the **AbsolutePosition** property can track the progress of a loop that enumerates all the records of a recordset. It uses the **CursorLocation** property to enable the **AbsolutePosition** property by setting the cursor to a client cursor.

```
Public Sub AbsolutePositionX()

Dim rstEmployees As ADODB.Recordset

Dim strCnn As String

Dim strMessage As String
```

```
' Open a recordset for the Employee table
' using a client cursor.
strCnn = "driver={SQL Server};server=srv;" & _
"uid=sa;pwd=;database=pubs"
Set rstEmployees = New ADODB.Recordset
' Use client cursor to enable AbsolutePosition property.
rstEmployees.CursorLocation = adUseClient
rstEmployees.Open "employee", strCnn, , , adCmdTable
' Enumerate Recordset.
Do While Not rstEmployees.EOF
' Display current record information.
strMessage = "Employee: " & rstEmployees!lName & vbCr & _
"(record " & rstEmployees.AbsolutePosition & _
" of " & rstEmployees.RecordCount & ")"
If MsgBox(strMessage, vbOKCancel) = vbCancel _
Then Exit Do
rstEmployees.MoveNext
Loop
rstEmployees.Close
End Sub
```

ADO Recordset Object ActiveConnection Property

Specifies to which **Connection** object the **Recordset** object currently belongs.

ActiveConnection Property Return Values (ADO Recordset Object)

Sets or returns a **String** containing the definition for a connection or an ADO Connection Object. Default is a **Null** object reference.

ActiveConnection Property Remarks (ADO Recordset Object)

Use the **ActiveConnection** property to determine the **Connection** object over which the specified **Command** object will execute or the specified recordset will be opened.

For open **Recordset** objects or for **Recordset** objects whose ADO Recordset Object Source Property is set to a valid ADO Command Object, the **ActiveConnection** property is read-only. Otherwise, it is read/write.

You can set this property to a valid **Connection** object or to a valid connection string. In this case, the provider creates a new **Connection** object using this definition and opens the

connection. Additionally, the provider may set this property to the new **Connection** object to give you a way to access the **Connection** object for extended error information or to execute other commands.

If you use the *ActiveConnection* argument of the ADO Recordset Object Open Method to open a **Recordset** object, the **ActiveConnection** property will inherit the value of the argument.

If you set the **Source** property of the **Recordset** object to a valid **Command** object variable, the **ActiveConnection** property of the recordset inherits the setting of the **Command** object's **ActiveConnection** property.

ActiveConnection Property Example (ADO Recordset Object)

This Visual Basic example uses the **ActiveConnection,** ADO Command Object CommandText Property**, CommandTimeout,** ADO Command Object CommandType Property**,** ADO Parameter Object Size Property**,** and ADO Parameter Object Direction Property properties to execute a stored procedure:

```
Public Sub ActiveConnectionX()

Dim cnn1 As ADODB.Connection

Dim cmdByRoyalty As ADODB.Command

Dim prmByRoyalty As ADODB.Parameter

Dim rstByRoyalty As ADODB.Recordset

Dim rstAuthors As ADODB.Recordset

Dim intRoyalty As Integer

Dim strAuthorID As String

Dim strCnn As String

` Define a command object for a stored procedure.

Set cnn1 = New ADODB.Connection

strCnn = "driver={SQL Server};server=srv;" & _

"uid=sa;pwd=;database=pubs"

cnn1.Open strCnn

Set cmdByRoyalty = New ADODB.Command

Set cmdByRoyalty.ActiveConnection = cnn1

cmdByRoyalty.CommandText = "byroyalty"

cmdByRoyalty.CommandType = adCmdStoredProc

cmdByRoyalty.CommandTimeout = 15

` Define the stored procedure's input parameter.

intRoyalty = Trim(InputBox( _
```

```
"Enter royalty:"))
Set prmByRoyalty = New ADODB.Parameter
prmByRoyalty.Type = adInteger
prmByRoyalty.Size = 3
prmByRoyalty.Direction = adParamInput
prmByRoyalty.Value = intRoyalty
cmdByRoyalty.Parameters.Append prmByRoyalty
` Create a recordset by executing the command.
Set rstByRoyalty = cmdByRoyalty.Execute()
` Open the Authors table to get author names for display.
Set rstAuthors = New ADODB.Recordset
rstAuthors.Open "authors", strCnn, , , adCmdTable
` Print current data in the recordset, adding
` author names from Authors table.
Debug.Print "Authors with " & intRoyalty & _
" percent royalty"
Do While Not rstByRoyalty.EOF
strAuthorID = rstByRoyalty!au_id
Debug.Print , rstByRoyalty!au_id & ", ";
rstAuthors.Filter = "au_id = '" & strAuthorID & "'"
Debug.Print rstAuthors!au_fname & " " & _
rstAuthors!au_lname
rstByRoyalty.MoveNext
Loop
rstByRoyalty.Close
rstAuthors.Close
cnn1.Close
End Sub
```

ADO Recordset Object BOF, EOF Properties

BOF indicates that the current record position is before the first record in a **Recordset** object.

EOF indicates that the current record position is after the last record in a **Recordset** object.

BOF, EOF Properties Return Values

The **BOF** and **EOF** properties return **Boolean** values.

BOF, EOF Properties Remarks

Use the **BOF** and **EOF** properties to determine whether a **Recordset** object contains records or whether you've gone beyond the limits of a **Recordset** object when you move from record to record.

The **BOF** property returns **True** (-1) if the current record position is before the first record and **False** (0) if the current record position is on or after the first record.

The **EOF** property returns **True** if the current record position is after the last record and **False** if the current record position is on or before the last record.

If either the **BOF** or **EOF** property is **True**, there is no current record.

If you open a **Recordset** object containing no records, the **BOF** and **EOF** properties are set to **True**, and the **Recordset** object's **RecordCount** property setting is zero. When you open a **Recordset** object that contains at least one record, the first record is the current record and the **BOF** and **EOF** properties are **False**.

If you delete the last remaining record in the **Recordset** object, the **BOF** and **EOF** properties may remain **False** until you attempt to reposition the current record.

This table shows which ADO Recordset Object Move Method methods are allowed with different combinations of the **BOF** and **EOF** properties:

	MoveFirst MoveLast	**MovePrevious Move < 0**	**Move 0**	**MoveNext Move > 0**
BOF = True, EOF = False	Allowed	Error	Error	Allowed
BOF=False EOF=True	Allowed	Allowed	Error	Error
Both True	Error	Error	Error	Error
Both False	Allowed	Allowed	Allowed	Allowed

Allowing a **Move** method doesn't guarantee that the method will successfully locate a record; it only means that calling the specified **Move** method won't generate an error.

The following table shows what happens to the **BOF** and **EOF** property settings when you call various **Move** methods but are unable to successfully locate a record.

	BOF	EOF
MoveFirst, MoveLast	Set to **True**	Set to **True**
Move 0	No change	No change
MovePrevious, Move < 0	Set to **True**	No change
MoveNext, Move > 0	No change	Set to **True**

BOF, EOF Properties Example

This Visual Basic example uses the **BOF** and **EOF** properties to display a message if a user tries to move past the first or last record of a recordset. It uses the ADO Recordset Object Bookmark Property to let the user flag a record in a recordset and return to it later.

```
Public Sub BOFX()
Dim rstPublishers As ADODB.Recordset
Dim strCnn As String
Dim strMessage As String
Dim intCommand As Integer
Dim varBookmark As Variant
` Open recordset with data from Publishers table.
strCnn = "driver={SQL Server};server=srv;" & _
"uid=sa;pwd=;database=pubs"
Set rstPublishers = New ADODB.Recordset
rstPublishers.CursorType = adOpenStatic
` Use client cursor to enable AbsolutePosition property.
rstPublishers.CursorLocation = adUseClient
rstPublishers.Open "SELECT pub_id, pub_name FROM publishers " & _
"ORDER BY pub_name", strCnn, , , adCmdText
rstPublishers.MoveFirst
Do While True
` Display information about current record
` and get user input.
strMessage = "Publisher: " & rstPublishers!pub_name & _
vbCr & "(record " & rstPublishers.AbsolutePosition & _
" of " & rstPublishers.RecordCount & ")" & vbCr & vbCr & _
"Enter command:" & vbCr & _
"[1 - next / 2 - previous /" & vbCr & _
```

```
"3 - set bookmark / 4 - go to bookmark]"
intCommand = Val(InputBox(strMessage))
Select Case intCommand
` Move forward or backward, trapping for BOF
` or EOF.
Case 1
rstPublishers.MoveNext
If rstPublishers.EOF Then
MsgBox "Moving past the last record." & _
vbCr & "Try again."
rstPublishers.MoveLast
End If
Case 2
rstPublishers.MovePrevious
If rstPublishers.BOF Then
MsgBox "Moving past the first record." &
_vbCr & "Try again."
rstPublishers.MoveFirst
End If
` Store the bookmark of the current record.
Case 3
varBookmark = rstPublishers.Bookmark
` Go to the record indicated by the stored
` bookmark.
Case 4
If IsEmpty(varBookmark) Then
MsgBox "No Bookmark set!"
Else
rstPublishers.Bookmark = varBookmark
End If
Case Else
Exit Do
```

```
End Select
Loop
rstPublishers.Close
End Sub
```

ADO Recordset Object Bookmark Property

Returns a bookmark that uniquely identifies the current record in a **Recordset** object or sets the current record in a **Recordset** object to the record identified by a valid bookmark.

Bookmark Property Return Values

Sets or returns a **Variant** expression that evaluates to a valid bookmark.

Bookmark Property Remarks

Use the **Bookmark** property to save the position of the current record and return to that record at any time. Bookmarks are available only in **Recordset** objects that support bookmark functionality.

When you open a **Recordset** object, each of its records has a unique bookmark. To save the bookmark for the current record, assign the value of the **Bookmark** property to a variable. To quickly return to that record at any time after moving to a different record, set the **Recordset** object's **Bookmark** property to the value of that variable.

The user may not be able to view the value of the bookmark. Also, users should not expect bookmarks to be directly comparable—two bookmarks that refer to the same record may have different values.

If you use the ADO Recordset Object Clone Method to create a copy of a **Recordset** object, the **Bookmark** property settings for the original and the duplicate **Recordset** objects are identical and you can use them interchangeably. However, you can't use bookmarks from different **Recordset** objects interchangeably, even if they were created from the same source or command.

Bookmark Property Examples

See the ADO Recordset Object BOF, EOF Properties.

ADO Recordset Object CacheSize Property

The number of records from a **Recordset** object that are cached locally in memory. *This property is not currently supported on UNIX.*

CacheSize Property Return Values

Sets or returns a **Long** value that must be greater than 0. Default is 1.

CacheSize Property Remarks

Use the **CacheSize** property to control how many records the provider keeps in its buffer and how many to retrieve at one time into local memory. For example, if the **CacheSize** is 10, after first

opening the **Recordset** object, the provider retrieves the first 10 records into local memory. As you move through the **Recordset** object, the provider returns the data from the local memory buffer. As soon as you move past the last record in the cache, the provider retrieves the next 10 records from the data source into the cache.

The value of this property can be adjusted during the life of the **Recordset** object, but changing this value only affects the number of records in the cache after subsequent retrievals from the data source. Changing the property value alone will not change the current contents of the cache.

If there are fewer records to retrieve than **CacheSize** specifies, the provider returns the remaining records; no error occurs.

A **CacheSize** setting of zero is not allowed and returns an error.

Records retrieved from the cache don't reflect concurrent changes that other users made to the source data. To force an update of all the cached data, use the ADO Recordset Object Resync Method.

CacheSize Property Example

This Visual Basic example uses the **CacheSize** property to show the difference in performance for an operation performed with and without a 30-record cache.

```
Public Sub CacheSizeX()

Dim rstRoySched As ADODB.Recordset

Dim strCnn As String

Dim sngStart As Single

Dim sngEnd As Single

Dim sngNoCache As Single

Dim sngCache As Single

Dim intLoop As Integer

Dim strTemp As String

` Open the RoySched table.

strCnn = "driver={SQL Server};server=srv;" & _

"uid=sa;pwd=;database=pubs"

Set rstRoySched = New ADODB.Recordset

rstRoySched.Open "roysched", strCnn, , , adCmdTable

` Enumerate the Recordset object twice and record

` the elapsed time.

sngStart = Timer

For intLoop = 1 To 2

rstRoySched.MoveFirst
```

```
Do While Not rstRoySched.EOF
' Execute a simple operation for the performance test.
strTemp = rstRoySched!title_id
rstRoySched.MoveNext
Loop
Next intLoop
sngEnd = Timer
sngNoCache = sngEnd - sngStart
' Cache records in groups of 30 records.
rstRoySched.MoveFirst
rstRoySched.CacheSize = 30
sngStart = Timer
` Enumerate the Recordset object twice and record
' the elapsed time.
For intLoop = 1 To 2
rstRoySched.MoveFirst
Do While Not rstRoySched.EOF
` Execute a simple operation for the
` performance test.
strTemp = rstRoySched!title_id
rstRoySched.MoveNext
Loop
Next intLoop
sngEnd = Timer
sngCache = sngEnd - sngStart
' Display performance results.
MsgBox "Caching Performance Results:" & vbCr & _
" No cache: " & Format(sngNoCache, _
"##0.000") & " seconds" & vbCr & _
" 30-record cache: " & Format(sngCache, _
"##0.000") & " seconds"
rstRoySched.Close
```

```
End Sub
```

ADO Recordset Object CursorLocation Property

Sets or returns the location of the cursor engine. *This property is read-only on UNIX.*

CursorLocation Property Return Values

Sets or returns a **Long** value that can be set to one of the following constants:

Constant	Description
adUseClient	Uses client-side cursors supplied by a local cursor library. Local cursor engines will often allow many features that driver-supplied cursors may not, so using this setting may provide an advantage with respect to features that will be enabled. For backward-compatibility, the synonym **adUseClientBatch** is also supported.
adUseServer	Default. Uses data-provider or driver-supplied cursors. These cursors are sometimes very flexible and allow for some additional sensitivity to reflecting changes that others make to the actual data source. However, some features of the Microsoft Client Cursor Provider (such as disassociated recordsets) cannot be simulated.

CursorLocation Property Remarks

This property allows you to choose between various cursor libraries accessible to the provider. Usually, you can choose between using a client-side cursor library or one that is located on the server.

This property setting only affects connections established after the property has been set. Changing the **CursorLocation** property has no effect on existing connections.

This property is read/write on a closed recordset, and read-only on an open recordset.

CursorLocation Property Example

See the **AbsolutePosition** property example.

ADO Recordset Object CursorType Property

The type of cursor used in a **Recordset** object.

CursorType Property Return Values

Sets or returns one of the following **CursorTypeEnum** values:

Constant	Description
adOpenForwardOnly	Forward-only cursor. Default. Identical to a static cursor except that you can only scroll forward through records. This improves performance in situations when you only need to make a single pass through a recordset.

adOpenKeyset	Keyset cursor. Like a dynamic cursor, except that you can't see records that other users add, although records that other users delete are inaccessible from your recordset. Data changes by other users are still visible.
adOpenDynamic	Dynamic cursor. Additions, changes, and deletions by other users are visible, and all types of movement through the recordset are allowed, except for bookmarks if the provider doesn't support them.
adOpenStatic	Static cursor. A static copy of a set of records that you can use to find data or generate reports. Additions, changes, or deletions by other users are not visible.

CursorType Property Remarks

Use the **CursorType** property to specify the type of cursor that should be used when opening the **Recordset** object. The **CursorType** property is read/write when the recordset is closed and read-only when it is open.

If a provider does not support the requested cursor type, the provider may return another cursor type. The **CursorType** property will change to match the actual cursor type in use when the recordset object is open. To verify specific functionality of the returned cursor, use the ADO Recordset Object Supports Method. After you close the recordset, the **CursorType** property reverts to its original setting.

The following chart shows the provider functionality (identified by **Supports** method constants) required for each cursor type.

CursorType	The Supports method must return True for these constants
adOpenForwardOnly	none
adOpenKeyset	adBookmark, adHoldRecords, adMovePrevious, adResync
adOpenDynamic	adMovePrevious
adOpenStatic	adBookmark, adHoldRecords, adMovePrevious, adResync

Note

Although **Supports**(**adUpdateBatch**) may be true for dynamic and forward-only cursors, for batch updates you should use either a keyset or static cursor. Set the ADO Recordset Object LockType Property to **adLockBatchOptimistic**, and set the **CursorLocation** property to **adUseClient** (or its synonym, **adUseClientBatch**) to enable the Microsoft Client Cursor Engine, which is required for batch updates.

CursorType Property Example

This Visual Basic example demonstrates setting the **CursorType** and **LockType** properties before opening a recordset. It also shows the value of the ADO Recordset Object EditMode

Property under various conditions. The **EditModeOutput** function is required for this procedure to run.

```
Public Sub EditModeX()
Dim cnn1 As ADODB.Connection
Dim rstEmployees As ADODB.Recordset
Dim strCnn As String
` Open recordset with data from Employee table.
Set cnn1 = New ADODB.Connection
strCnn = "driver={SQL Server};server=srv;" & _
"uid=sa;pwd=;database=pubs"
cnn1.Open strCnn
Set rstEmployees = New ADODB.Recordset
Set rstEmployees.ActiveConnection = cnn1
rstEmployees.CursorType = adOpenKeyset
rstEmployees.LockType = adLockBatchOptimistic
rstEmployees.Open "employee", , , , adCmdTable
` Show the EditMode property under different editing
` states.
rstEmployees.AddNew
rstEmployees!emp_id = "T-T55555M"
rstEmployees!fname = "temp_fname"
rstEmployees!lname = "temp_lname"
EditModeOutput "After AddNew:", rstEmployees.EditMode
rstEmployees.UpdateBatch
EditModeOutput "After UpdateBatch:", rstEmployees.EditMode
rstEmployees!fname = "test"
EditModeOutput "After Edit:", rstEmployees.EditMode
rstEmployees.Close
` Delete new record because this is a demonstration.
cnn1.Execute "DELETE FROM employee WHERE emp_id = 'T-T55555M'"
End Sub
Public Function EditModeOutput(strTemp As String, _
intEditMode As Integer)
```

```
` Print report based on the value of the EditMode
` property.
Debug.Print strTemp
Debug.Print " EditMode = ";
Select Case intEditMode
Case adEditNone
Debug.Print "adEditNone"
Case adEditInProgress
Debug.Print "adEditInProgress"
Case adEditAdd
Debug.Print "adEditAdd"
End Select
End Function
```

ADO Recordset Object EditMode Property

The editing status of the current record.

EditMode Property Return Values

Returns one of the following **EditModeEnum** values:

Constant	Description
adEditNone	No editing operation is in progress.
adEditInProgress	The data in the current record has been modified but not yet saved.
adEditAdd	The **AddNew** method has been invoked and the current record in the copy buffer is a new record that hasn't been saved in the database.

EditMode Property Remarks

ADO maintains an editing buffer associated with the current record. This property indicates whether changes have been made to this buffer, or whether a new record has been created. Use the **EditMode** property to determine the editing status of the current record. You can test for pending changes if an editing process has been interrupted and determine whether you need to use the ADO Recordset Object Update Method or ADO Recordset Object CancelUpdate Method.

See the ADO Recordset Object AddNew Method for a more detailed description of the **EditMode** property under different editing conditions.

EditMode Property Example

See the ADO Recordset Object CursorType Property example.

ADO Recordset Object Filter Property

A filter for data in a recordset.

Filter Property Return Values

Sets or returns a **Variant** value, which can contain one of the following:

Criteria string

A string made up of one or more individual clauses concatenated with **AND** or **OR** operators.

Array of bookmarks

An array of unique bookmark values that point to records in the **Recordset** object. *This return value is not currently supported on UNIX.*

*One of the following **FilterGroupEnum** values:*

Constant	Description
adFilterNone	Removes the current filter and restores all records to view.
adFilterPendingRecords	Enables you to view only records that have changed but have not yet been sent to the server. Only applicable for batch update mode. *Not currently supported on UNIX.*
adFilterAffectedRecords	Enables you to view only records affected by the last **Delete**, **Resync**, **UpdateBatch**, or **CancelBatch** call. *Not currently supported on UNIX.*
adFilterFetchedRecords	Enables you to view records in the current cache, that is, the results of the last call to retrieve records from the database. *Not currently supported on UNIX.*

Filter Property Remarks

Use the **Filter** property to selectively screen out records in a **Recordset** object. The filtered recordset becomes the current cursor. This affects other properties such as **AbsolutePosition, AbsolutePage, RecordCount,** and ADO Recordset Object PageCount Property that return values based on the current cursor, since setting the **Filter** property to a specific value will move the current record to the first record that satisfies the new value.

On UNIX systems the **Filter** property is implemented for **Recordset** objects whose source is a SELECT query. Setting the **Filter** property will resubmit the query with the criteria string AND'd with the WHERE clause.

The criteria string is made up of clauses in the form *FieldName-Operator-Value* (for example, `"LastName = 'Smith'"`). You can create compound clauses by concatenating individual clauses with **AND** (for example, `"LastName = 'Smith' AND FirstName =`

'John'") or **OR** (for example, "LastName = 'Smith' OR LastName = 'Jones'"). Use the following guidelines for criteria strings:

FieldName

Must be a valid field name from the recordset. If the field name contains spaces, you must enclose the name in square brackets.

Operator

Must be one of the following: <, >, <=, >=, <>, =, **LIKE**.

Value

The value with which you will compare the field values (for example, 'Smith', #8/24/95#, 12.345 or $50.00). Use single quotes with strings and pound signs (#) with dates. For numbers, you can use decimal points, dollar signs, and scientific notation. If *Operator* is **LIKE**, *Value* can use wildcards. Only the asterisk (*) and percent sign (%) wildcards are allowed, and they must be the last character in the string. *Value* may not be **Null**.

There is no precedence between **AND** and **OR**. Clauses can be grouped within parentheses. However, you cannot group clauses joined by an **OR** and then join the group to another clause with an **AND**, like this:

```
(LastName = 'Smith' OR LastName = 'Jones') AND FirstName = 'John'
```

Instead, you would construct this filter as:

```
(LastName = 'Smith' AND FirstName = 'John') OR
(LastName = 'Jones' AND FirstName = 'John')
```

In a **LIKE** clause, you can use a wildcard at the beginning and end of the pattern (for example, LastName Like '*mit*'), or only at the end of the pattern (for example, LastName Like 'Smit*').

The filter constants make it easier to resolve individual record conflicts during batch update mode by allowing you to view, for example, only those records that were affected during the last ADO Recordset Object UpdateBatch Method call.

Setting the **Filter** property itself may fail because of a conflict with the underlying data (for example, a record has already been deleted by another user); in such a case, the provider returns warnings to the ADO Errors Collection but does not halt program execution. A run-time error occurs only if there are conflicts on all the requested records. Use the ADO Recordset Object Status Property to locate records with conflicts.

Setting the **Filter** property to a zero-length string ("") has the same effect as using the **adFilterNone** constant.

Whenever the **Filter** property is set, the current record position moves to the first record in the filtered subset of records in the recordset. Similarly, when the **Filter** property is cleared, the current record position moves to the first record in the recordset.

See the ADO Recordset Object Bookmark Property for an explanation of bookmark values from which you can build an array to use with the **Filter** property.

Filter Property Example

This Visual Basic example uses the **Filter** property to open a new recordset based on a specified condition applied to an existing recordset. It uses the **RecordCount** property to show the number of records in the two recordsets. The **FilterField** function is required for this procedure to run.

```
Public Sub FilterX()
Dim rstPublishers As ADODB.Recordset
Dim rstPublishersCountry As ADODB.Recordset
Dim strCnn As String
Dim intPublisherCount As Integer
Dim strCountry As String
Dim strMessage As String
` Open recordset with data from Publishers table.
strCnn = "driver={SQL Server};server=srv;" & _
"uid=sa;pwd=;database=pubs"
Set rstPublishers = New ADODB.Recordset
rstPublishers.CursorType = adOpenStatic
rstPublishers.Open "publishers", strCnn, , , adCmdTable
` Populate the Recordset.
intPublisherCount = rstPublishers.RecordCount
` Get user input.
strCountry = Trim(InputBox( _
"Enter a country to filter on:"))
If strCountry <> "" Then
` Open a filtered Recordset object.
Set rstPublishersCountry = _
FilterField(rstPublishers, "Country", strCountry)
If rstPublishersCountry.RecordCount = 0 Then
MsgBox "No publishers from that country."
Else
` Print number of records for the original
` Recordset object and the filtered Recordset
` object.
strMessage = "Orders in original recordset: " & _
```

```
vbCr & intPublisherCount & vbCr & _
"Orders in filtered recordset (Country = '" & _
strCountry & "'): " & vbCr & _
rstPublishersCountry.RecordCount
MsgBox strMessage
End If
rstPublishersCountry.Close
End If
End Sub
Public Function FilterField(rstTemp As ADODB.Recordset, _
strField As String, strFilter As String) As ADODB.Recordset
` Set a filter on the specified Recordset object and then
` open a new Recordset object.
rstTemp.Filter = strField & " = '" & strFilter & "'"
Set FilterField = rstTemp
End Function
```

Note

When you know the data you want to select, it's usually more efficient to open a recordset with an SQL statement. This example shows how you can create just one recordset and obtain records from a particular country.

```
Public Sub FilterX2()
Dim rstPublishers As ADODB.Recordset
Dim strCnn As String
` Open recordset with data from Publishers table.
strCnn = "driver={SQL Server};server=srv;" & _
"uid=sa;pwd=;database=pubs"
Set rstPublishers = New ADODB.Recordset
rstPublishers.CursorType = adOpenStatic
rstPublishers.Open "SELECT * FROM publishers " & _
"WHERE Country = 'USA'", strCnn, , , adCmdText


` Print current data in recordset.
```

```
rstPublishers.MoveFirst
Do While Not rstPublishers.EOF
Debug.Print rstPublishers!pub_name & ", " & _
rstPublishers!country
rstPublishers.MoveNext
Loop
rstPublishers.Close
End Sub
```

ADO Recordset Object LockType Property

The type of locks placed on records during editing.

LockType Property Return Values

Sets or returns one of the following **LockTypeEnum** values:

Constant	Description
adLockReadOnly	Default. Read-only; the data cannot be modified.
adLockPessimistic	Pessimistic locking, record by record. The provider does what is necessary to ensure successful editing of the records, usually by locking records at the data source immediately upon editing.
adLockOptimistic	Optimistic locking, record by record. The provider uses optimistic locking, locking records only when you call the **Update** method.
adLockBatchOptimistic	Optimistic batch updates. Required for batch update mode as opposed to immediate update mode.

LockType Property Remarks

Set the **LockType** property before opening a recordset to specify what type of locking the provider should use when opening it. Read the property to return the type of locking in use on an open **Recordset** object. The **LockType** property is read/write when the recordset is closed and read-only when it is open.

Providers may not support all lock types. If a provider cannot support the requested **LockType** setting, it will substitute another type of locking. To determine the actual locking functionality available in a **Recordset** object, use the ADO Recordset Object Supports Method with **adUpdate** and **adUpdateBatch**.

LockType Property Example

See the ADO Recordset Object CursorType Property example.

ADO Recordset Object MarshalOptions Property

Indicates which records are to be marshaled back to the server. *This is a client-side only property.*

MarshalOptions Property Return Values

Sets or returns a **Long** value that can be one of the following constants:

Constant	Description
adMarshalAll	Default. All rows are returned to the server.
adMarshalModifiedOnly	Only modified rows are returned to the server.

MarshalOptions Property Remarks

When using a client-side (ADOR) recordset, records that have been modified on the client are written back to the middle-tier or Web server through a technique called *marshaling*, the process of packaging and sending interface method parameters across thread or process boundaries. Setting the **MarshalOptions** property can improve performance when modified remote data is marshaled for updating back to the middle-tier or Web server.

Remote Data Service Usage: This property is only used on a client-side (ADOR) recordset.

MarshalOptions Property Example

This Visual Basic example uses the **MarshalOptions** property to specify what rows are sent back to the server—All Rows or only Modified Rows.

```
Public Sub MarshalOptionsX()

Dim rstEmployees As ADODB.Recordset

Dim strCnn As String

Dim strOldFirst As String

Dim strOldLast As String

Dim strMessage As String

Dim strMarshalAll As String

Dim strMarshalModified As String

` Open recordset with names from Employee table.

strCnn = "driver={SQL Server};server=srv;" & _

"uid=sa;pwd=;database=pubs"

Set rstEmployees = New ADODB.Recordset

rstEmployees.CursorType = adOpenKeyset

rstEmployees.LockType = adLockOptimistic

rstEmployees.CursorLocation = adUseClient

rstEmployees.Open "SELECT fname, lname " & _
```

```
"FROM Employee ORDER BY lname", strCnn, , , adCmdText
` Store original data.
strOldFirst = rstEmployees!fname
strOldLast = rstEmployees!lname
` Change data in edit buffer.
rstEmployees!fname = "Linda"
rstEmployees!lname = "Kobara"
` Show contents of buffer and get user input.
strMessage = "Edit in progress:" & vbCr & _
" Original data = " & strOldFirst & " " & _
strOldLast & vbCr & " Data in buffer = " & _
rstEmployees!fname & " " & rstEmployees!lname & vbCr & vbCr & _
"Use Update to replace the original data with " & _
"the buffered data in the Recordset?"
strMarshalAll = "Would you like to send all the rows " & _
"in the recordset back to the server?"
strMarshalModified = "Would you like to send only " & _
"modified rows back to the server?"
If MsgBox(strMessage, vbYesNo) = vbYes Then
If MsgBox(strMarshalAll, vbYesNo) = vbYes Then
rstEmployees.MarshalOptions = adMarshalAll
rstEmployees.Update
ElseIf MsgBox(strMarshalModified, vbYesNo) = vbYes Then
rstEmployees.MarshalOptions = adMarshalModifiedOnly
rstEmployees.Update
End If
End If
` Show the resulting data.
MsgBox "Data in recordset = " & rstEmployees!fname & " " & _
rstEmployees!lname
` Restore original data because this is a demonstration.
If Not (strOldFirst = rstEmployees!fname And _
```

```
strOldLast = rstEmployees!lname) Then
rstEmployees!fname = strOldFirst
rstEmployees!lname = strOldLast
rstEmployees.Update
End If
rstEmployees.Close
End Sub
```

ADO Recordset Object MaxRecords Property

The maximum number of records to return to a recordset from a query.

MaxRecords Property Return Values

Sets or returns a **Long** value. Default is zero (no limit).

MaxRecords Property Remarks

Use the **MaxRecords** property to limit the number of records the provider returns from the data source. The default setting of this property is zero, which means the provider returns all requested records. The **MaxRecords** property is read/write when the recordset is closed and read-only when it is open.

MaxRecords Property Example

This Visual Basic example uses the **MaxRecords** property to open a recordset containing the 10 most expensive titles in the ***Titles*** table.

```
Public Sub MaxRecordsX()
Dim rstTemp As ADODB.Recordset
Dim strCnn As String
` Open recordset containing the 10 most expensive
` titles in the Titles table.
strCnn = "driver={SQL Server};server=srv;" & _
"uid=sa;pwd=;database=pubs"
Set rstTemp = New ADODB.Recordset
rstTemp.MaxRecords = 10
rstTemp.Open "SELECT Title, Price FROM Titles " & _
"ORDER BY Price DESC", strCnn, , , adCmdText
` Display the contents of the recordset.
Debug.Print "Top Ten Titles by Price:"
```

```
Do While Not rstTemp.EOF
Debug.Print " " & rstTemp!Title & " - " & rstTemp!Price
rstTemp.MoveNext
Loop
rstTemp.Close
End Sub
```

ADO Recordset Object PageCount Property

The number of pages of data the **Recordset** object contains.

PageCount Property Return Values

Returns a **Long** value.

PageCount Property Remarks

Use the **PageCount** property to determine how many pages of data are in the **Recordset** object. *Pages* are groups of records whose size equals the ADO Recordset Object PageSize Property setting. Even if the last page is incomplete, because there are fewer records than the **PageSize** value, it counts as an additional page in the **PageCount** value. If the **Recordset** object does not support this property, the value will be -1 to indicate that the **PageCount** is indeterminable.

See the **PageSize** and **AbsolutePage** properties for more on page functionality.

PageCount Property Example

See the **AbsolutePage** example.

ADO Recordset Object PageSize Property

The number of records that constitute one page in the recordset.

PageSize Property Return Values (ADO Recordset Object)[0]

Sets or returns a **Long** value, the number of records on a page. Default is 10.

PageSize Property Remarks (ADO Recordset Object)

Use the **PageSize** property to determine how many records make up a logical page of data. Establishing a page size allows you to use the **AbsolutePage** property to move to the first record of a particular page. This is useful in Web-server scenarios when you want to allow the user to page through data, viewing a certain number of records at a time.

This property can be set at any time, and its value will be used for calculating where the first record of a particular page is.

PageSize Property Example (ADO Recordset Object)

See the **AbsolutePage** property example.

ADO Recordset Object State Property

Describes the current state of an object.

State Property Return Values (ADO Recordset Object)

Sets or returns a **Long** value that can be one of the following constants:

Constant	Description
AdStateClosed	Default. The object is closed.
AdStateOpen	The object is open.

State Property Remarks (ADO Recordset Object)

You can use the **State** property to determine the current state of a given object at any time.

ADO Recordset Object Status Property

Indicates the status of the current record with respect to batch updates or other bulk operations.

Status Property Return Values (ADO Recordset Object)

Returns a sum of one or more of the following **RecordStatusEnum** values:

Constant	Description
adRecOK	The record was successfully updated.
adRecNew	The record is new.
adRecModified	The record was modified.
adRecDeleted	The record was deleted.
adRecUnmodified	The record was not modified.
adRecInvalid	The record was not saved because its bookmark is invalid.
adRecMultipleChanges	The record was not saved because it would have affected multiple records.
adRecPendingChanges	The record was not saved because it refers to a pending insert.
adRecCanceled	The record was not saved because the operation was canceled.
adRecCantRelease	The new record was not saved because of existing record locks.
adRecConcurrencyViolation	The record was not saved because optimistic concurrency was in use.
adRecIntegrityViolation	The record was not saved because the user

	violated integrity constraints.
adRecMaxChangesExceeded	The record was not saved because there were too many pending changes.
adRecObjectOpen	The record was not saved because of a conflict with an open storage object.
adRecOutOfMemory	The record was not saved because the computer has run out of memory.
adRecPermissionDenied	The record was not saved because the user has insufficient permissions.
adRecSchemaViolation	The record was not saved because it violates the structure of the underlying database.
adRecDBDeleted	The record has already been deleted from the data source.

Status Property Remarks (ADO Recordset Object)

Use the **Status** property to see what changes are pending for records modified during batch updating. You can also use the **Status** property to view the status of records that fail during bulk operations such as when you call the ADO Recordset Object Resync Method, ADO Recordset Object UpdateBatch Method, or ADO Recordset Object CancelBatch Method methods on a **Recordset** object, or set the ADO Recordset Object Filter Property on a **Recordset** object to an array of bookmarks. With this property, you can determine how a given record failed and resolve it accordingly.

Status Property Example (ADO Recordset Object)

This example uses the **Status** property to display which records have been modified in a batch operation before a batch update has occurred.

```
Public Sub StatusX()

Dim rstTitles As ADODB.Recordset

Dim strCnn As String

` Open recordset for batch update.

strCnn = "driver={SQL Server};server=srv;" & _

"uid=sa;pwd=;database=pubs"

Set rstTitles = New ADODB.Recordset

rstTitles.CursorType = adOpenKeyset

rstTitles.LockType = adLockBatchOptimistic

rstTitles.Open "titles", strCnn, , , adCmdTable

` Change the type of psychology titles.

Do Until rstTitles.EOF
```

```
If Trim(rstTitles!Type) = "psychology" Then
rstTitles!Type = "self_help"
End If
rstTitles.MoveNext
Loop
` Display Title ID and status.
rstTitles.MoveFirst
Do Until rstTitles.EOF
If rstTitles.Status = adRecModified Then
Debug.Print rstTitles!title_id & " - Modified"
Else
Debug.Print rstTitles!title_id
End If
rstTitles.MoveNext
Loop
` Cancel the update because this is a demonstration.
rstTitles.CancelBatch
rstTitles.Close
End Sub
```

ADO Recordset Object Source Property

The source for the data in a **Recordset** object (**Command** object, SQL statement, table name, or stored procedure).

Source Property Return Values (ADO Recordset Object)

Sets a **String** value or **Command** object reference; returns only a **String** value.

Source Property Remarks (ADO Recordset Object)

Use the **Source** property to specify a data source for a **Recordset** object using one of the following: an ADO Command Object variable, an SQL statement, a stored procedure, or a table name. The **Source** property is read/write for closed **Recordset** objects and read-only for open **Recordset** objects.

If you set the **Source** property to a **Command** object, the **ActiveConnection** property of the **Recordset** object will inherit the value of the **ActiveConnection** property for the specified **Command** object. However, reading the **Source** property does not return a **Command** object;

instead, it returns the **CommandText** property of the **Command** object to which you set the **Source** property.

If the **Source** property is an SQL statement, a stored procedure, or a table name, you can optimize performance by passing the appropriate *Options* argument with the ADO Recordset Object Open Method call.

Source Property Example (ADO Recordset Object)

This Visual Basic example demonstrates the **Source** property by opening three **Recordset** objects based on different data sources.

```
Public Sub SourceX()
Dim cnn1 As ADODB.Connection
Dim rstTitles As ADODB.Recordset
Dim rstPublishers As ADODB.Recordset
Dim rstTitlesPublishers As ADODB.Recordset
Dim cmdSQL As ADODB.Command
Dim strCnn As String
Dim strSQL As String
` Open a connection.
Set cnn1 = New ADODB.Connection
strCnn = "driver={SQL Server};server=srv;" & _
"uid=sa;pwd=;database=pubs"
cnn1.Open strCnn
` Open a recordset based on a command object.
Set cmdSQL = New ADODB.Command
Set cmdSQL.ActiveConnection = cnn1
cmdSQL.CommandText = "Select title, type, pubdate " & _
"FROM titles ORDER BY title"
Set rstTitles = cmdSQL.Execute()
` Open a recordset based on a table.
Set rstPublishers = New ADODB.Recordset
rstPublishers.Open "publishers", strCnn, , , adCmdTable
` Open a recordset based on an SQL string.
Set rstTitlesPublishers = New ADODB.Recordset
strSQL = "SELECT title_ID AS TitleID, title AS Title, " & _
```

```
"publishers.pub_id AS PubID, pub_name AS PubName " & _
"FROM publishers INNER JOIN titles " & _
"ON publishers.pub_id = titles.pub_id " & _
"ORDER BY Title"
rstTitlesPublishers.Open strSQL, strCnn, , , adCmdText
` Use the Source property to display the source of each recordset.
MsgBox "rstTitles source: " & vbCr & _
rstTitles.Source & vbCr & vbCr & _
"rstPublishers source: " & vbCr & _
rstPublishers.Source & vbCr & vbCr & _
"rstTitlesPublishers source: " & vbCr & _
rstTitlesPublishers.Source
rstTitles.Close
rstPublishers.Close
rstTitlesPublishers.Close
cnn1.Close
End Sub
```

ADO Recordset Object RecordCount Property

The current number of records in a **Recordset** object.

RecordCount Property Return Values

Returns a **Long** value.

RecordCount Property Remarks

Use the **RecordCount** property to find out how many records are in a **Recordset** object. The property returns -1 when ADO cannot determine the number of records. Reading the **RecordCount** property on a closed recordset causes an error.

If the **Recordset** object supports approximate positioning or bookmarks—that is, ADO Recordset Object Supports Method **(adApproxPosition)** or **Supports (adBookmark)**, respectively, returns **True**—this value will be the exact number of records in the recordset regardless of whether it has been fully populated. If the **Recordset** object does not support approximate positioning, this property may be a significant drain on resources because all records will have to be retrieved and counted to return an accurate **RecordCount** value.

RecordCount Property Example

This Visual Basic example uses the **Filter** property to open a new recordset based on a specified condition applied to an existing recordset. It uses the **RecordCount** property to show the number of records in the two recordsets. The **FilterField** function is required for this procedure to run.

```
Public Sub FilterX()
Dim rstPublishers As ADODB.Recordset
Dim rstPublishersCountry As ADODB.Recordset
Dim strCnn As String
Dim intPublisherCount As Integer
Dim strCountry As String
Dim strMessage As String
` Open recordset with data from Publishers table.
strCnn = "driver={SQL Server};server=srv;" & _
"uid=sa;pwd=;database=pubs"
Set rstPublishers = New ADODB.Recordset
rstPublishers.CursorType = adOpenStatic
rstPublishers.Open "publishers", strCnn, , , adCmdTable
` Populate the Recordset.
intPublisherCount = rstPublishers.RecordCount
` Get user input.
strCountry = Trim(InputBox( _
"Enter a country to filter on:"))
If strCountry <> "" Then
` Open a filtered Recordset object.
Set rstPublishersCountry = _
FilterField(rstPublishers, "Country", strCountry)
If rstPublishersCountry.RecordCount = 0 Then
MsgBox "No publishers from that country."
Else
` Print number of records for the original
` Recordset object and the filtered Recordset
` object.
strMessage = "Orders in original recordset: " & _
vbCr & intPublisherCount & vbCr & _
```

```
"Orders in filtered recordset (Country = '" & _
strCountry & "'): " & vbCr & _
rstPublishersCountry.RecordCount
MsgBox strMessage
End If
rstPublishersCountry.Close
End If
End Sub
Public Function FilterField(rstTemp As ADODB.Recordset, _
strField As String, strFilter As String) As ADODB.Recordset
` Set a filter on the specified Recordset object and then
` open a new Recordset object.
rstTemp.Filter = strField & " = '" & strFilter & "'"
Set FilterField = rstTemp
End Function
```

Note

When you know the data you want to select, it's usually more efficient to open a recordset with an SQL statement. This example shows how you can create just one recordset and obtain records from a particular country.

```
Public Sub FilterX2()
Dim rstPublishers As ADODB.Recordset
Dim strCnn As String
` Open recordset with data from Publishers table.
strCnn = "driver={SQL Server};server=srv;" & _
"uid=sa;pwd=;database=pubs"
Set rstPublishers = New ADODB.Recordset
rstPublishers.CursorType = adOpenStatic
rstPublishers.Open "SELECT * FROM publishers " & _
"WHERE Country = 'USA'", strCnn, , , adCmdText
` Print current data in recordset.
```

```
rstPublishers.MoveFirst

Do While Not rstPublishers.EOF

Debug.Print rstPublishers!pub_name & ", " & _

rstPublishers!country

rstPublishers.MoveNext

Loop

rstPublishers.Close

End Sub
```

ADO Collections

Collections

ADO Errors Collection	Contains all stored **Error** objects, all of which pertain to a single operation involving ADO.
ADO Fields Collection	Contains all stored **Field** objects of a **Recordset** object.
ADO Parameters Collection	Contains all the **Parameter** objects of a **Command** object.
ADO Properties Collection	Contains all the **Property** objects for the specific instance of an object. *This collection is not currently supported on UNIX.*

Methods

ADO Collections Append Method	Appends a new object to the **Parameters** collection.
ADO Collections Clear Method	Clears the contents of an **Errors** collection.
ADO Collections Delete Method	Deletes an object from the **Parameters** collection.
ADO Collections Item Method	Returns a specific member of a collection by name or ordinal number.
ADO Collections Refresh Method	Updates the objects in a collection to reflect objects available from and specific to the provider.

Properties

ADO Collections Count Property	The number of objects in a collection.

ADO Errors Collection

The **Errors** collection contains all stored ADO Error Object objects created in response to a single failure involving the provider.

ADO Errors Collection Remarks

Any operation involving ADO objects can generate one or more provider errors. As each error occurs, one or more ADO Error Object objects may be placed in the **Errors** collection of the ADO Connection Object. When another ADO operation generates an error, the **Errors** collection is cleared, and the new set of **Error** objects may be placed in the **Errors** collection.

Each **Error** object represents a specific provider error, not an ADO error. ADO errors are exposed to the run-time exception-handling mechanism. For example, in Microsoft Visual Basic, the occurrence of an ADO-specific error will trigger an **On Error** event and appear in the **Err** object.

ADO operations that don't generate an error have no effect on the **Errors** collection. Use the ADO Collections Clear Method to manually clear the **Errors** collection.

The set of **Error** objects in the **Errors** collection describes all errors that occurred in response to a single statement. Enumerating the specific errors in the **Errors** collection enables your error-handling routines to more precisely determine the cause and origin of an error, and take appropriate steps to recover.

Some properties and methods return warnings that appear as **Error** objects in the **Errors** collection but do not halt a program's execution. Before you call the ADO Recordset Object Resync Method**,** ADO Recordset Object UpdateBatch Method**,** or ADO Recordset Object CancelBatch Method methods on an ADO Recordset Object, or before you set the ADO Recordset Object Filter Property on a **Recordset** object, call the **Clear** method on the **Errors** collection so that you can read the Count Property of the **Errors** collection to test for returned warnings.

Note

> See the ADO Error Object for a more detailed explanation of the way a single ADO operation can generate multiple errors.

ADO Fields Collection

The **Fields** collection contains all the **Field** objects of a **Recordset** object.

ADO Fields Collection Remarks

An ADO Recordset Object has a **Fields** collection made up of ADO Field Object objects. Each **Field** object corresponds to a column in the recordset. You can populate the **Fields** collection before opening the recordset by calling the ADO Collections Refresh Method on the collection.

Note

> See the ADO Field Object for a more detailed explanation of how to use **Field** objects.

ADO Parameters Collection

The **Parameters** collection contains all the **Parameter** objects of a **Command** object.

ADO Parameters Collection Remarks

An ADO Command Object has a **Parameters** collection made up of ADO Parameter Object objects. Using the ADO Collections Refresh Method on a **Command** object's **Parameters** collection retrieves provider parameter information for the stored procedure or parameterized query specified in the **Command** object. Some providers do not support stored procedure calls or parameterized queries; calling the **Refresh** method on the **Parameters** collection when using such a provider will return an error.

If you have not defined your own **Parameter** objects and you access the **Parameters** collection before calling the **Refresh** method, ADO will automatically call the method and populate the collection for you.

You can minimize calls to the provider to improve performance if you know the properties of the parameters associated with the stored procedure or parameterized query you wish to call. Use the **CreateParameter** method to create **Parameter** objects with the appropriate property settings and use the ADO Collections Append Method to add them to the **Parameters** collection. This lets you set and return parameter values without having to call the provider for the parameter information. If you are writing to a provider that does not supply parameter information, you must manually populate the **Parameters** collection using this method to be able to use parameters at all. Use the ADO Collections Delete Method to remove **Parameter** objects from the **Parameters** collection if necessary.

ADO Properties Collection

The **Properties** collection contains all the **Property** objects for a specific instance of an object. *The **Property** collection is not currently supported on UNIX.*

ADO Properties Collection Remarks

Some ADO objects have a **Properties** collection made up of ADO Property Object objects. Each **Property** object corresponds to a characteristic of the ADO object specific to the provider.

Note

> See the ADO Property Object topic for a more detailed explanation of how to use **Property** objects.

ADO Collections Methods

ADO Collections Append Method

Appends an object to a collection.

Append Method Applies To

ADO Parameters Collection

Append Method Syntax

```
collection.Append object
```

Append Method Parameters

object

An object variable representing the object to be appended.

Append Method Remarks

Use the **Append** method on a collection to add an object to that collection. This method is available only on the **Parameters** collection of a ADO Command Object. You must set the ADO Parameter Object Type Property of an ADO Parameter Object before appending it to the **Parameters** collection. If you select a variable-length data type, you must also set the ADO Parameter Object Size Property to a value greater than zero.

By describing the parameter yourself, you can minimize calls to the provider and consequently improve performance when using stored procedures or parameterized queries. However, you must know the properties of the parameters associated with the stored procedure or parameterized query you wish to call. Use the **CreateParameter** method to create **Parameter** objects with the appropriate property settings and use the **Append** method to add them to the **Parameters** collection. This lets you set and return parameter values without having to call the provider for the parameter information. If you are writing to a provider that does not supply parameter information, you must manually populate the **Parameters** collection using this method to be able to use parameters at all.

Append Method Examples

This Visual Basic example uses the **Append** and **CreateParameter** methods to execute a stored procedure with an input parameter.

```
Public Sub AppendX()

Dim cnn1 As ADODB.Connection

Dim cmdByRoyalty As ADODB.Command

Dim prmByRoyalty As ADODB.Parameter

Dim rstByRoyalty As ADODB.Recordset

Dim rstAuthors As ADODB.Recordset

Dim intRoyalty As Integer

Dim strAuthorID As String

Dim strCnn As String

` Open connection.

Set cnn1 = New ADODB.Connection
```

```
strCnn = "driver={SQL Server};server=srv;" & _
"uid=sa;pwd=;database=pubs"
cnn1.Open strCnn
cnn1.CursorLocation = adUseClient
` Open command object with one parameter.
Set cmdByRoyalty = New ADODB.Command
cmdByRoyalty.CommandText = "byroyalty"
cmdByRoyalty.CommandType = adCmdStoredProc
` Get parameter value and append parameter.
intRoyalty = Trim(InputBox("Enter royalty:"))
Set prmByRoyalty = cmdByRoyalty.CreateParameter("percentage", _
adInteger, adParamInput)
cmdByRoyalty.Parameters.Append prmByRoyalty
prmByRoyalty.Value = intRoyalty
` Create recordset by executing the command.
Set cmdByRoyalty.ActiveConnection = cnn1
Set rstByRoyalty = cmdByRoyalty.Execute
` Open the Authors table to display author names.
Set rstAuthors = New ADODB.Recordset
rstAuthors.Open "authors", cnn1, , , adCmdTable
` Print current data in the recordset, adding
` author names from Authors table.
Debug.Print "Authors with " & intRoyalty & " percent royalty"
Do While Not rstByRoyalty.EOF
strAuthorID = rstByRoyalty!au_id
Debug.Print " " & rstByRoyalty!au_id & ", ";
rstAuthors.Filter = "au_id = '" & strAuthorID & "'"
Debug.Print rstAuthors!au_fname & " " & rstAuthors!au_lname
rstByRoyalty.MoveNext
Loop
rstByRoyalty.Close
rstAuthors.Close
```

```
cnn1.Close

End Sub
```

ADO Collections Clear Method

Removes all of the objects in a collection.

Clear Method Applies To

ADO Errors Collection

Clear Method Syntax

```
Errors.Clear
```

Clear Method Remarks

Use the **Clear** method on the **Errors** collection to remove all existing ADO Error Object objects from the collection. When an error occurs, ADO automatically clears the **Errors** collection and fills it with **Error** objects based on the new error. However, some properties and methods return warnings that appear as **Error** objects in the **Errors** collection but do not halt a program's execution. Before you call the ADO Recordset Object Resync Method**,** ADO Recordset Object UpdateBatch Method**,** or ADO Recordset Object CancelBatch Method methods on an ADO Recordset Object or before you set the ADO Recordset Object Filter Property on a **Recordset** object, call the **Clear** method on the **Errors** collection. Doing so enables you to read the ADO Collections Count Property of the **Errors** collection to test for returned warnings as a result of these specific calls.

Clear Method Examples

See the ADO Command Object Execute Method.

ADO Collections Delete Method

Deletes an object from the **Parameters** collection.

Delete Method Applies To

ADO Parameters Collection

Delete Method Syntax

```
object.Parameters.Delete ( Index )
```

Delete Method Parameters

object

A **Command** object.

Index

A **Variant** that evaluates either to the name or to the ordinal number of an object in a collection.

Delete Method Remarks

Using the **Delete** method on a **Parameters** collection lets you remove one of the objects in the collection. This method is available only on the **Parameters** collection of an ADO Command Object. You must the use ADO Parameter Object object's ADO Parameter Object Name Property or its collection index when calling the **Delete** method; an object variable is not a valid argument.

ADO Collections Item Method

Returns a specific member of a collection by name or ordinal number.

Item Method Applies To

ADO Errors Collection**,** ADO Fields Collection**,** ADO Parameters Collection**,** ADO Properties Collection

Item Method Syntax

```
Set object = collection.Item ( Index )
```

Item Method Parameters

object

Object reference created.

Index

A **Variant** that evaluates either to the name or to the ordinal number of an object in a collection.

Item Method Return Values

Returns an object reference.

Item Method Remarks

Use the **Item** method to return a specific object in a collection. If the method cannot find an object in the collection corresponding to the *Index* argument, an error occurs. Also, some collections don't support named objects; for these collections, you must use ordinal number references.

The **Item** method is the default method for all collections; therefore, the following syntax forms are interchangeable:

```
collection.Item (Index)

collection (Index)
```

ADO Collections Refresh Method

Updates the objects in a collection to reflect objects available from and specific to the provider.

Refresh Method Applies To

ADO Fields Collection**,** ADO Parameters Collection**,** ADO Properties Collection

Refresh Method Syntax

```
collection.Refresh
```

Refresh Method Parameters Collection

Using the **Refresh** method on a ADO Command Object object's **Parameters** collection retrieves provider-side parameter information for the stored procedure or parameterized query specified in the **Command** object. The collection will be empty for providers that do not support stored procedure calls or parameterized queries.

You should set the **ActiveConnection** property of the **Command** object to a valid ADO Connection Object, the ADO Command Object CommandText Property to a valid command, and the ADO Command Object CommandType Property to **adCmdStoredProc** before calling the ADO Collections Refresh Method.

If you access the **Parameters** collection before calling the **Refresh** method, ADO will automatically call the method and populate the collection for you.

Note

> If you use the **Refresh** method to obtain parameter information from the provider and it returns one or more variable-length data type ADO Parameter Object objects, ADO may allocate memory for the parameters based on their maximum potential size, which will cause an error during execution. You should explicitly set the ADO Parameter Object Size Property for these parameters before calling the ADO Command Object Execute Method to prevent errors.

Refresh Method Fields Collection

Using the **Refresh** method on the **Fields** collection has no visible effect. To retrieve changes from the underlying database structure, you must use either the ADO Recordset Object Requery Method or, if the **Recordset** object does not support bookmarks, the ADO Recordset Object MoveFirst, MoveLast, MoveNext, MovePrevious Methods method.

Refresh Method Properties Collection

Using the **Refresh** method on a **Properties** collection of some objects populates the collection with the dynamic properties the provider exposes. These properties provide information about functionality specific to the provider beyond the built-in properties ADO supports.

The **Refresh** method accomplishes different tasks depending on the collection from which you call it.

Refresh Method Example

This Visual Basic example demonstrates using the **Refresh** method to refresh the **Parameters** collection for a stored procedure **Command** object.

```
Public Sub RefreshX()

Dim cnn1 As ADODB.Connection

Dim cmdByRoyalty As ADODB.Command
```

```
Dim rstByRoyalty As ADODB.Recordset
Dim rstAuthors As ADODB.Recordset
Dim intRoyalty As Integer
Dim strAuthorID As String
Dim strCnn As String
' Open connection.
Set cnn1 = New ADODB.Connection
strCnn = "driver={SQL Server};server=srv;" & _
"uid=sa;pwd=;database=pubs"
cnn1.Open strCnn
' Open a command object for a stored procedure
' with one parameter.
Set cmdByRoyalty = New ADODB.Command
Set cmdByRoyalty.ActiveConnection = cnn1
cmdByRoyalty.CommandText = "byroyalty"
cmdByRoyalty.CommandType = adCmdStoredProc
cmdByRoyalty.Parameters.Refresh
' Get paramater value and execute the command,
' storing the results in a recordset.
intRoyalty = Trim(InputBox("Enter royalty:"))
cmdByRoyalty.Parameters(1) = intRoyalty
Set rstByRoyalty = cmdByRoyalty.Execute()
` Open the Authors table to get author names for display.
Set rstAuthors = New ADODB.Recordset
rstAuthors.Open "authors", cnn1, , , adCmdTable
' Print current data in the recordset, adding
' author names from Authors table.
Debug.Print "Authors with " & intRoyalty & " percent royalty"
Do While Not rstByRoyalty.EOF
strAuthorID = rstByRoyalty!au_id
Debug.Print " " & rstByRoyalty!au_id & ", ";
rstAuthors.Filter = "au_id = '" & strAuthorID & "'"
```

```
Debug.Print rstAuthors!au_fname & " " & _
rstAuthors!au_lname
rstByRoyalty.MoveNext
Loop
rstByRoyalty.Close
rstAuthors.Close
cnn1.Close
End Sub
```

ADO Collections Properties

ADO Collections Count Property

The number of objects in a collection.

Count Property Applies To

ADO Errors Collection**,** ADO Fields Collection**,** ADO Parameters Collection**,** ADO Properties Collection

Count Property Return Values

Returns a **Long** value.

Count Property Remarks

Use the **Count** property to determine how many objects are in a given collection.

Because numbering for members of a collection begins with zero, you should always code loops starting with the zero member and ending with the value of the **Count** property minus one. If you are using Visual Basic and want to loop through the members of a collection without checking the **Count** property, use the **For Each...Next** command.

If the **Count** property is zero, there are no objects in the collection.

Count Property Example

This Visual Basic example demonstrates the **Count** property with two collections in the ***Employee*** database. The property obtains the number of objects in each collection, and sets the upper limit for loops that enumerate these collections. Another way to enumerate these collections without using the **Count** property would be to use **For Each...Next** statements.

```
Public Sub CountX()
Dim rstEmployees As ADODB.Recordset
Dim strCnn As String
Dim intloop As Integer
```

```
' Open recordset with data from Employee table.
strCnn = "driver={SQL Server};server=srv;" & _
"uid=sa;pwd=;database=pubs"
Set rstEmployees = New ADODB.Recordset
rstEmployees.Open "employee", strCnn, , , adCmdTable
' Print information about Fields collection.
Debug.Print rstEmployees.Fields.Count & _
" Fields in Employee"
For intloop = 0 To rstEmployees.Fields.Count - 1
Debug.Print " " & rstEmployees.Fields(intloop).Name
Next intloop
' Print information about Properties collection.
Debug.Print rstEmployees.Properties.Count & _
" Properties in Employee"
For intloop = 0 To rstEmployees.Properties.Count - 1 Debug.Print " "
& rstEmployees.Properties(intloop).Name
Next intloop
rstEmployees.Close
End Sub
```

ASP Built-in Objects Reference

Sun Chili!Soft ASP provides five built-in or intrinsic objects for the Active Server Pages (ASP) framework: **Request**, **Response**, **Application**, **Server**, and **Session**. Built-in objects are objects that are included on all ASP pages; they do not need to be created before they can be used. This section discusses the following ASP built-in objects:

ASP Built-in Object	Description
ASP Application Object	Stores information (variables and objects) needed for all users of a particular application. Information stored in the **Application** object persists for the lifetime of the application.
ASP Request Object	Provides access to values passed to the server by the client.
ASP Response Object	Controls the output from an ASP script to the requesting client.
ASP Server Object	Provides access to methods and properties on the server. These methods and properties typically serve as utility functions.

ASP Session Object	Stores information (variables and objects) needed for a particular user session. Information stored in the **Session** object is not discarded when the user jumps between pages in the application; instead information persists for the entire user session.

Built-in objects simplify development by solving Web-protocol programming issues. For example, built-in objects can provide mechanisms to process data sent using the Hypertext Transfer Protocol (HTTP). HTTP is a stateless technology, and the server cannot track the location of users in an application or the use of an application. By using the built-in **Session** object, an application (or object) can handle session management tasks.

Note

ASP scripts provided in the examples are assumed to be enclosed in script delimiters. The <%, %>, <SCRIPT>, and </SCRIPT> delimiters are generally not shown.

See also:

Using Sun Chili!Soft ASP Built-in Objects in "Chapter 4: Building a Sun Chili!Soft ASP Application"

ASP Application Object

The ASP **Application** object shares information among all users of a given application. An ASP-based application is defined as all the .asp files in a virtual directory and associated sub-directories. Because the **Application** object can be shared by more than one user, **Lock** and **Unlock** methods are provided to ensure that multiple users do not try to alter a property simultaneously.

Syntax: ASP Application Object

```
Application.method
```

ASP Application Object Methods

ASP Application Object Lock Method	Prevents script from modifying object properties.
ASP Application Object Unlock Method	Enables script to modify object properties after execution of the **Lock** method.

ASP Application Object Events

Application_OnStart	Runs when an ASP page belonging to the application is accessed for the first time.
Application_OnEnd	Runs when the Web server is shut down.

Values can be stored in the **Application** object. These values are available throughout the application and have application scope.

Objects can be created within the **Application_OnStart** script and assigned to the **Application** object. You cannot, however, store a built-in object in the **Application** object. Each of the following lines will return an error:

```
Set Application("var1") = Session

Set Application("var2") = Request

Set Application("var3") = Response

Set Application("var4") = Server

Set Application("var5") = Application
```

Before you store an object in the **Application** object, you must know what threading model it uses. Only objects marked as both free and apartment-threaded can be stored in the **Application** object.

The **Application** object is implemented as a collection. If you store an array in an **Application** object, you should not attempt to alter elements of the stored array directly. For example, the following script does not work:

```
Application("StoredArray") (3) = "new value"
```

Instead of storing the value "`new value`" in `StoredArray(3)` the value is stored in the **Application** collection, overwriting any information stored at `Application(3)`.

Note

> It is strongly recommended that if you store an array in the **Application** object that you retrieve a copy of the array before retrieving or changing any of the elements of the array. When you are done making changes to the array, store the array back into the **Application** object to save changes. This is demonstrated in the following examples.

ASP Application Object Examples

You can store different types of variables:

```
Application("greeting") = "Welcome to My Web World"

Application("num") = 25
```

You must use the **Set** keyword when storing objects:

```
Set Application("Obj1") = Server.CreateObject("MyComponent")
```

You can use methods and properties on subsequent ASP pages by using the following:

```
Application("Obj1").MyObjMethod
```

As an alternative, you can extract a local copy of the object:

```
Set MyLocalObj1 = Application("Obj1")

MyLocalObj.MyObjMethod
```

The next example demonstrates using an application variable called `NumVisits` to store the number of times a particular page has been accessed. The **Lock** method is called to ensure only

the current client can access or alter `NumVisits`. Calling the **Unlock** method then enables other clients to access the application object.

```
Application.Lock

Application("NumVisits") = Application("NumVisits") + 1

Application.Unlock

This application page has been visited <%= Application("NumVisits")
%> times!
```

The next three examples demonstrate storing and manipulating an array in the **Application** object. The **Lock** and **Unlock** methods are used to control access to the **Application** object.

```
Application.Lock

Application("StoredArray") = MyArray

Application.Unlock
```

To retrieve the array from the **Application** object and modify its elements:

```
LocalArray = Application("StoredArray")

LocalArray(0) = "Hello"

LocalArray(1) = "there"
```

Next you need to restore the array in the **Application** object. This overwrites the values in `StoredArray` with new values.

```
Application.Lock

Application("StoredArray") = LocalArray

Application.Unlock
```

ASP Application Object Methods

ASP Application Object Lock Method

The **Lock** method blocks other clients from modifying variables stored in the **Application** object, ensuring that only one client at a time can alter or access the application variables. If you do not call the **Unlock** method explicitly, the server unlocks the locked **Application** object when the script ends or times out.

Syntax: ASP Application Object Lock Method

```
Application.Lock
```

ASP Application Object Unlock Method

The **Unlock** method enables other clients (via an ASP page) to modify the variables stored in the **Application** object after it has been locked using the **Lock** method. If you do not call the **Unlock**

method explicitly, the server unlocks the locked **Application** object when the script ends or times out.

Syntax: ASP Application Object Unlock Method

```
Application.Unlock
```

ASP Request Object

The **Request** object retrieves the values that the browser passed to the server during an HTTP request.

Syntax: ASP Request Object

```
Request.[collection | property | method] (variable)
```

ASP Request Object Collections

ASP Request Object Cookies Collection	The value of cookies sent in the HTTP request.
ASP Request Object Form Collection	The values of form elements sent in the HTTP request body.
ASP Request Object QueryString Collection	The value of variables in the HTTP query string.
ASP Request Object ServerVariables Collection	The value of predetermined environment variables.

Note

Due to widely differing Web-server support for client-side certificates, Sun Chili!Soft ASP does not implement the **ClientCertificate** collection.

ASP Request Object Properties

ASP Request Object TotalBytes Property	The total number of bytes the client is sending in the body of the request.

ASP Request Object Methods

ASP Request Object BinaryRead Method	Retrieves data sent to the server from the client as part of a POST request.

Variable parameters are strings that identify the item to be retrieved from a collection or a value to be passed to a property or method. For more information about the *variable* parameter, see the individual collection descriptions. If the specified variable is not in one of the collections, the **Request** object returns `EMPTY`.

All variables can be accessed directly by calling `Request("variable")` without a collection name. In this case, the Web server searches the collections in the following order:

- **QueryString**

- **Form**
- **Cookies**
- **ServerVariables**

If the same variable exists in more than one collection, the first one encountered will be used. It is strongly recommended that you use the collection name. For example, instead of **Request.**(AUTH_USER) use **Request.**ServerVariables(AUTH_USER).

ASP Request Object Collections

ASP Request Object Cookies Collection

The **Cookies** collection allows you to retrieve the values of the cookies sent in an HTTP request.

Syntax: ASP Request Object Cookies Collection

```
Request.Cookies(cookie)[(key)|.attribute]
```

Parameters: ASP Request Object Cookies Collection

cookie

Specifies the cookie whose value should be received.

key

An optional parameter used to retrieve subkey values from cookie dictionaries.

attribute

Specifies information about the cookie itself. The attribute value can be:

Name	Value
HasKeys	Read-only. Specifies whether the cookie contains keys.

Remarks: ASP Request Object Cookies Collection

Access the subkeys of a cookie dictionary by including a value for *key*. If a cookie dictionary is accessed without specifying a key, all of the keys are returned as a single query string. For example, if **MyCookie** has two keys, First and Second, and you do not specify either of these keys in a call to **Request.Cookies**, the following string is returned.

```
First=firstkeyvalue&Second=secondkeyvalue
```

If two cookies with the same name are sent by the client browser, **Request.Cookies** returns the one with the deeper path structure. For example, if two cookies had the same name but one had a path attribute of /www/ and the other of /www/home/, the client browser would send both cookies to the /www/home/ directory, but **Request.Cookies** would only return the second cookie.

To determine whether a cookie is a cookie dictionary (whether the cookie has keys), use the following script.

```
<%= Request.Cookies("myCookie").HasKeys %>
```

If `myCookie` is a cookie dictionary, the preceding value evaluates to `TRUE`; otherwise, it evaluates to `FALSE`.

You can use an iterator to cycle through all the cookies in the **Cookie** collection, or all the keys in a cookie. However, iterating through keys on a cookie that does not have keys will not produce any output. You can avoid this situation by first checking to see whether a cookie has keys by using the **HasKeys** attribute.

Examples: ASP Request Object Cookies Collection

The first example shows how to print the entire cookie collection:

```
<%
'Print out the entire cookie collection.
For Each cookie in Request.Cookies
  If Not cookie.HasKeys Then
    'Print out the cookie string
%>
    <%= cookie %> = <%= Request.Cookies(cookie) %>
<%
  Else
    'Print out the cookie collection
    For Each key in Request.Cookies(cookie) %>
      <%= cookie %> (<%= key %>) = <%= Request.Cookies(cookie)(key)
%>
<%
    Next
  End If
Next
%>
```

The next example prints the value of a cookie variable called "myCookie":

```
<%= Request.Cookies("myCookie") %>
```

ASP Request Object Form Collection

The **Form** collection retrieves the values of form elements posted to the HTTP request body by a form using the POST method.

Syntax: ASP Request Object Form Collection

Request.Form(*parameter***)**[**(***index***)**|**.Count**]

Parameters: ASP Request Object Form Collection

parameter

Specifies the name of the form element from which the collection is to retrieve values.

index

An optional parameter that enables you to access one of multiple values for a parameter. It can be any integer in the range 1 to *Count*.

Remarks: ASP Request Object Form Collection

The **Form** collection is indexed by the names of the parameters in the request body. The value of **Request.Form(***parameter***)** is an array of all of the values of *parameter* that occur in the request body. You can determine the number of values of a parameter by calling **Request.Form(***parameter***).Count**. If a parameter does not have multiple values associated with it, the count is 1. If the parameter is not bound, the count is 0.

To reference a single value of a form element that has multiple values, you must specify a value for the index. The *index* parameter may be any number between 1 and **Request.Form(***parameter***).Count**. If you reference one of multiple form parameters without specifying a value for *index*, the data is returned as a comma-delimited string.

When you use parameters with **Request.Form**, the Web server parses the HTTP request body and returns the specified data. If your application requires unparsed data from the form, you can access it by calling **Request.Form** without any parameters.

Examples: ASP Request Object Form Collection

In this example an iterator is used to loop through all the data values in a form request. Assume that a user fills out a form by specifying two values (Chocolate and Butterscotch) for the **FavoriteFlavor** parameter. The following script will retrieve these values:

```
For Each item In Request.Form("FavoriteFlavor")
Response.Write item & "<BR>"
Next
```

This displays the following:

```
Chocolate
Butterscotch
```

The same output can be generated with a **For…Next** loop, as shown in the following script:

```
For I = 1 To Request.Form("FavoriteFlavor").Count
Response.Write Request.Form("FavoriteFlavor")(I) & "<BR>"
Next
```

This iterator can display the parameter name, as shown in the following script.

```
<% For Each x In Request.Form %>
Request.Form( <%= x %> ) = <%= Request.Form(x) %> <BR>
```

```
<% Next %>
```

This displays the following:

```
FavoriteFlavor = Chocolate
FavoriteFlavor = Butterscotch
```

The next example uses the following form to solicit information from a user:

```
<FORM ACTION = "/scripts/submit.asp" METHOD = "post">
<P>Your first name: <INPUT NAME = "firstname" SIZE = 48>
<P>What is your favorite ice cream flavor: <SELECT NAME = "flavor">
<OPTION>Vanilla
<OPTION>Strawberry
<OPTION>Chocolate
<OPTION>Rocky Road</SELECT>
<p><INPUT TYPE = SUBMIT>
</FORM>
```

From that form, the following request body might be sent to the client:

```
firstname=James&flavor=Rocky+Road
```

The following script can then be used:

```
Welcome, <%= Request.Form("firstname") %>.
Your favorite flavor is <%= Request.Form("flavor") %>.
The unparsed form data is: <%= Request.Form %>
This displays the following:
"Welcome, James. Your favorite flavor is Rocky Road."
```

The unparsed form data is: `firstname=James&flavor=Rocky+Road`

ASP Request Object QueryString Collection

The **QueryString** collection retrieves the values of the variables in the HTTP query string; that is, it retrieves the values encoded after the question mark (?) in an HTTP request. For example, it parses the values sent by a form using the GET method.

Syntax: ASP Request Object QueryString Collection

```
Request.QueryString(variable)[(index)|.Count]
```

Parameters: ASP Request Object QueryString Collection

variable

Specifies the name of the variable in the HTTP query string to retrieve.

ASP Request Object QueryString Collection Index

An optional parameter that enables you to retrieve one of multiple values for *variable.* It can be any integer value in the range 1 to **Request.QueryString(***variable***).Count**.

Remarks: ASP Request Object QueryString Collection

The **QueryString** collection is a parsed version of the QUERY_STRING variable in the **ServerVariables** collection. It enables you to retrieve the QUERY_STRING variables by name. The value of **Request.QueryString(***parameter***)** is an array of all of the values of *parameter* that occur in QUERY_STRING. You can determine the number of values of a parameter by calling **Request.QueryString(***parameter***).Count**. If a variable does not have multiple data sets associated with it, the count is 1. If the variable is not found, the count is 0.

To reference a **QueryString** variable in one of multiple data sets, you specify a value for *index.* The *index* parameter may be any value between 1 and **Request.QueryString(***variable***).Count**. If you reference one of multiple **QueryString** variables without specifying a value for *index,* the data is returned as a comma-delimited string.

When you use parameters with **Request.QueryString**, the server parses the parameters sent to the request and returns the specified data. If your application requires unparsed **QueryString** data, you can retrieve it by calling **Request.QueryString** without any parameters.

Examples: ASP Request Object QueryString Collection

You can use an iterator to loop through all the data values in a query string. For example, if the following request is sent:

```
http://NAMES.ASP?Q=Fred&Q=Sally
```

and NAMES.ASP contained the following script:

```
For Each item In Request.QueryString("Q")
   Response.Write item & "<BR>"
Next
```

NAMES.ASP would display the following:

```
Fred
Sally
```

Instead of using **For Each**, you can loop through data values in a query string using the **Count** variable:

```
For I = 1 To Request.QueryString("Q").Count
Response.Write Request.QueryString("Q")(I) & "<BR>"
Next
```

The following client request:

```
/scripts/directory-lookup.asp?name=fred&age=22
```

results in the QUERY_STRING value:

```
name=fred&age=22.
```

The **QueryString** collection would then contain two members, `name` and `age`.

```
Welcome, <%= Request.QueryString("name") %>.

Your age is <%= Request.QueryString("age") %>.
```

This script displays:

```
"Welcome, Fred. Your age is 22."
```

ASP Request Object ServerVariables Collection

The **ServerVariables** collection retrieves the values of environment variables.

Syntax: Request Object ServerVariables Collection

Request.ServerVariables(*variable***)**

Parameters: Request Object ServerVariables Collection

variable

This specifies the name of the server environment variable to retrieve. It can be one of the values from the following table:

Value	Description
ALL_RAW	All headers in raw form as sent by the client.
APPL_MD_PATH*	Retrieves the metabase path for the application.
APPL_PHYSICAL_PATH	Retrieves the physical path corresponding to the metabase path.
ASP_VERSION	Version number of the Sun Chili!Soft ASP server.
ASP_VERSION_MAJOR	The major version number of the Sun Chili!Soft ASP server.
ASP_VERSION_MINOR	The minor version number of the Sun Chili!Soft ASP server.
ASP_OS	The operating system the server is running on.
ASP_LICENSE	License information for the Sun Chili!Soft ASP server.
AUTH_PASSWORD	The password corresponding to REMOTE_USER as supplied by the client.
AUTH_TYPE	If the server supports user authentication and the script is protected, this is the protocol-specific authentication method used to validate the user.
AUTH_USER	Raw authenticated user name.
CERT_COOKIE*	Unique ID for the client certificate, returned as a string.
CERT_FLAGS*	bit0 is set to 1 if the client certificate is present. bit1 is set to 1 if the Certifying Authority of the client certificate is invalid (not in the list of recognized CA on the server).
CERT_ISSUER*	Issuer field of the client certificate.

CERT_KEYSIZE*	Number of bits in the Secure Sockets Layer connection key size, for example, 128.
CERT_SECRETKEYSIZE*	Number of bits in the server certificate private key, for example 1024.
CERT_SERIALNUMBER*	Serial number field of the client certificate.
CERT_SERVER_ISSUER*	Issuer field of the server certificate.
CERT_SERVER_SUBJECT*	Subject field of the server certificate.
CERT_SUBJECT*	Subject field of the client certificate.
CONTENT_LENGTH	The length of content as given by the client.
CONTENT_TYPE	The data type of the content in queries that have attached information, such as HTTP GET, POST, and PUT.
GATEWAY_INTERFACE	The revision of the CGI specification used by the server. Format: CGI/*revision*.
HTTP_<HeaderName>	The value stored in the header *HeaderName.* Any header other than those listed in this table must be prefixed by "HTTP_" for the **ServerVariables** collection to retrieve its value. The server interprets any underscore (_) characters in *HeaderName* as dashes in the actual header. For example, if you specify HTTP_MY_HEADER, the server searches for MY-HEADER.
HTTPS	Returns "on" if the request came in through a secure channel or "off" if the request is for a non-secure channel.
HTTPS_KEYSIZE	Number of bits in Secure Sockets Layer key size, for example, 128.
HTTPS_SECRET_KEYSIZE	Number of bits in the server certificate private key, for example, 1024.
HTTPS_SERVER_ISSUER	Issuer field of the server certificate.
HTTPS_SERVER_SUBJECT	Subject field of the server certificate.
INSTANCE_ID	The ID for the instance in textual format. If the instance ID is 1, it appears as a string. Under IIS you can use this variable to retrieve the ID of the Web-server instance (in the metabase) to which the request belongs.
INSTANCE_META_PATH*	The metabase path for the instance of IIS that responds to the request.
LOCAL_ADDR	Returns the Server Address on which the request came in. This is important on multi-homed machines where there can be multiple IP addresses bound to a machine and you want to find out which address the request used.
LOGON_USER	The Windows account the client user is logged into.

Note

Not supported by Sun Chili!Soft ASP for UNIX.

PATH_INFO	The extra path information, as given by the client. Scripts can be accessed by using their virtual path and the PATH_INFO server variable. If this information comes from a URL, it is decoded by the server before it is passed to the script.
PATH_TRANSLATED	A translated version of PATH_INFO that takes the path and performs any virtual to physical mapping.
QUERY_STRING	Query information in the string following the question mark (?) in the HTTP request.
REMOTE_ADDR	The IP address of the remote host making the request.
REMOTE_HOST	The name of the host making the request. If the server does not have this information, it will set REMOTE_ADDR and leave this empty.
REMOTE_USER	If the server supports user authentication and the script is protected, this is the username by which the user is authenticated.
REQUEST_METHOD	The method used to make the request. For HTTP, this would be GET, HEAD, POST, etc.
SCRIPT_NAME	A virtual path to the script being executed. This is used for self-referencing URLs.
SERVER_NAME	The server's host name, DNS alias, or IP address as it would appear in self-referencing URLs.
SERVER_PORT	The port number to which the request was sent.
SERVER_PORT_SECURE	A string that contains either 0 or 1. If the request is being handled on the secure port, then this will be 1; otherwise it will be 0.
SERVER_PROTOCOL	The name and revision of the information protocol. Format: *protocol/revision.*
SERVER_SOFTWARE	The name and version of the server software answering the request and running the gateway. Format: *name/version.*
URL	Gives the base portion of the URL.

*These server variables are only valid when running Sun Chili!Soft ASP with Microsoft Internet Information Server. When using other Web servers they will always be empty.

Remarks: Request Object ServerVariables Collection

If a client sends a header other than those specified in the preceding table, you can retrieve the value of the header by prefixing the header name with "HTTP_" in the call to **Request.ServerVariables**. For example, if the client sent the following header:

```
SomeNewHeader:SomeNewValue
```

you could retrieve `SomeNewValue` by using the following:

```
<% Request.ServerVariables("HTTP_SomeNewHeader") %>
```

Examples: Request Object ServerVariables Collection

You can use an iterator to loop through each server variable name. For example, the following script prints all of the server variables in a table:

```
<TABLE>
<TR><TH>Server Variable</TH><TH>Value</TH></TR>
<% for each name in Request.ServerVariables %>
<TR>
   <TD><%= name %></TD>
   <TD><%= Request.ServerVariables(name) %></TD>
</TR>
<% Next %>
</TABLE>
```

The following example demonstrates using **Request.ServerVariables** to insert the name of a server into a hyperlink:

```
<A HREF="http://<%= Request.ServerVariables("SERVER_NAME")%>
   /scripts/MyPage.asp">Link to MyPage.asp</A>
```

ASP Request Object Methods

ASP Request Object BinaryRead Method

The **BinaryRead** method reads information sent from the client to the server as part of a POST request. The data is returned as a **SafeArray** that contains information about the dimensions of the array.

Syntax: ASP Request Object BinaryRead Method

```
variant = Request.BinaryRead(count)
```

Parameters: ASP Request Object BinaryRead Method

variant

An array of unsigned bytes returned by this method.

count

Before the read, the number of bytes to read from the client. After execution, the actual number of bytes successfully read from the client. The number of bytes that will be read is less than or equal to **Request.TotalBytes**.

Remarks: ASP Request Object BinaryRead Method

The **BinaryRead** method is used to read the raw data sent by a POST request. This provides low-level access as opposed to the formatted data provided by the **Request.Form** collection. Once you have used the **BinaryRead** method, any call to a variable in the **Request.Form** collection

will cause an error. Conversely, calling **BinaryRead** after accessing the **Request.Form** collection will also cause an error. Remember, if you access a variable in the **Request** object without specifying a collection, the **Request.Form** collection may be accessed, bringing this rule into force.

Examples: ASP Request Object BinaryRead Method

The following example uses the **BinaryRead** method to place the contents of a **Request** object into a safe array.

```
<%
dim bytecount
dim binread
bytecount = Request.TotalBytes
binread = Request.BinaryRead(bytecount)
%>
```

ASP Request Object Properties

ASP Request Object TotalBytes Property

The **TotalBytes** property contains the total number of bytes sent by the client in the body of the request. The property is read-only.

Syntax: ASP Request Object TotalBytes Property

```
Counter = Request.TotalBytes
```

Parameters: ASP Request Object TotalBytes Property

counter

A variable to hold the total number of bytes the client sent in the request.

Examples: ASP Request Object TotalBytes Property

The following example sets a variable equal to the total number of bytes included in a request object:

```
<%
dim bytecount
bytecount = Request.TotalBytes
%>
```

ASP Response Object

The **Response** object controls sending output to the browser.

Syntax: ASP Response Object

```
Response.collection | property | method
```

ASP Response Object Collections

ASP Response Object Cookies Collection	You can use this collection to set cookie values to send to the client browser.

ASP Response Object Properties

ASP Response Object Buffer Property	Indicates whether to buffer page output.
ASP Response Object CacheControl Property	Determines if proxy servers are allowed to cache the output generated by ASP.
ASP Response Object Charset Property	Appends the name of the character set to the content-type header.
ASP Response Object ContentType Property	Specifies the HTTP content type for the response.
ASP Response Object Expires Property	Specifies the length of time until the page cached on a browser expires.
ASP Response Object ExpiresAbsolute Property	Specifies the date and time a page cached on a browser expires.
ASP Response Object IsClientConnected Property	Indicates if the client is still connected to the server.
ASP Response Object PICS Property	Adds the value of a PICS label to the pics-label field of the response header.
ASP Response Object Status Property	The value of the status line returned by the server.

ASP Response Object Methods

ASP Response Object AddHeader Method	Set the HTML header *name* to *value*.
ASP Response Object AppendToLog Method	Adds a string to the end of the Web server log entry for this request.
ASP Response Object BinaryWrite Method	Writes the given information to the current HTTP output without any character set conversion.
ASP Response Object Clear Method	Erases any buffered HTML output.
ASP Response Object End Method	Stops processing the .asp file and returns the current results.
ASP Response Object Flush Method	Sends any buffered HTML output immediately.
ASP Response Object Redirect Method	Sends a redirect message to the browser, causing it to attempt to connect to a different URL.
ASP Response Object Write Method	Writes a variable to the current HTML output as a string.

ASP Response Object Collections

ASP Response Object Cookies Collection

The **Cookies** collection sets the value of a cookie. If a specified cookie does not exist, it is created. If it does exist, the cookie takes on the new value and the old value is discarded.

Syntax: ASP Response Object Cookies Collection

```
Response.Cookies(cookie) [(key) | .attribute] = value
```

Parameters: ASP Response Object Cookies Collection

cookie

The name of the cookie.

key

Optional. If *key* is specified, the cookie is a dictionary and *key* is set to *value*.

attribute

Specific information about the cookie itself. The attribute can be one of the following:

Attribute	Description
Expires	The date on which the cookie expires. This attribute must be set to a date later than the current date to store the cookie on the client disk after the current session ends. Write-only.
HasKeys	Indicates that the cookie has keys. Read-only.
Path	If set, the cookie is only sent to requests on this path. If the attribute is not set, the application path is used. Write-only.
Secure	Indicates that the cookie is secure. Write-only.

value

Specifies the value to assign to *key* or *attribute*.

Remarks: ASP Response Object Cookies Collection

If a cookie with keys is created, as in the following script:

```
Response.Cookies("myCookie")("type1") = "sugar"

Response.Cookies("myCookie")("type2") = "ginger snap"
```

the following header is sent:

```
SET-COOKIE:MYCOOKIE=TYPE1=sugar&TYPE2=ginger+snap
```

Any subsequent assignment to `myCookie` that does not include a key would destroy `type1` and `type2`. The following example discards the values `type1` and `type2` and replaces them with the value "`chocolate chip`":

```
Response.Cookies("myCookie") = "chocolate chip"
```

Conversely, calling a cookie with a key destroys any non-key values the cookie might contain. The following code will discard the value "`chocolate chip`" and insert the key value instead:

```
Response.Cookies("myCookie") ("NewType") = "peanut butter"
```

To check to see if a cookie has key values, use the following:

```
Response.Cookies("myCookie").HasKeys
```

If `myCookie` is a dictionary and has keys, the previous script will evaluate to TRUE, otherwise it will be FALSE.

You can use an iterator to set cookie attributes. The following example sets all the cookies in a collection to expire on Dec. 31, 1999:

```
<%
For each cookie in Response.Cookies
   Response.Cookies(cookie).ExpiresAbsolute = #Dec. 31, 1999#
Next
%>
```

An iterator can also be used to set the values of all the cookies in a collection, or all the keys in a cookie. However, when using an iterator to retrieve cookie values, the cookies must have keys or the iterator will not execute. Use the **HasKeys** property to check to see whether a cookie has any keys. This is demonstrated in the following example.

```
<%
If Not cookie.HasKeys Then
   'Set the value of the cookie
   Response.Cookies(cookie) = ""
Else
   'Set the value for each key in the cookie collection
   For Each key in Response.Cookies(cookie)
      Response.Cookies(cookie)(key) = ""
   Next key
%>
```

Examples: ASP Response Object Cookies Collection

The following example shows how you can set cookie values and their attributes:

```
<%
Response.Cookies("Type") = "Chocolate Chip"
Response.Cookies("Type").Expires = "July 31, 1997"
```

```
Response.Cookies("Type").Domain = "msn.com"
Response.Cookies("Type").Path = "/www/home/"
Response.Cookies("Type").Secure = FALSE
%>
```

ASP Response Object Methods

ASP Response Object AddHeader Method

The **AddHeader** method adds an HTML header with a specified value. This method always adds a new HTTP header to the response. It will not replace an existing header of the same name. Once a header has been added, it cannot be removed.

This method is for advanced use only. If another **Response** method will provide the functionality you require, it is recommended that you use that method instead.

Syntax: ASP Response Object AddHeader Method

Response.AddHeader *name, value*

Parameters: ASP Response Object AddHeader Method

name

The name of the header variable.

value

The value assigned to the header variable.

Remarks: ASP Response Object AddHeader Method

To avoid any name ambiguity, the name of the header should not contain any underscores (_). The **Request.ServerVariables** collection interprets underscores as dashes in the header name. The following script causes a search for a header named "My-Header":

```
<% Request.ServerVariables("HTTP_MY_HEADER") %>
```

Because HTTP protocol requires that all headers be sent before content, you must call the **AddHeader** method in your script before any output (such as that generated by HTML code or the ASP Response Object Write Method) is sent to the client. The exception to this rule is when the ASP Response Object Buffer Property is set to **True**. If the output is buffered, you can call the **AddHeader** method at any point in the script, as long as it precedes any calls to the ASP Response Object Flush Method. Otherwise, the call to **AddHeader** will generate a run-time error.

The following two examples illustrate this. In the first example, the page is not buffered. The script works, however, because the **AddHeader** method is called before the server sends the Web page to the client. If the order were reversed, the call to the **AddHeader** method would generate a run-time error.

```
<% Response.AddHeader "WARNING", "Error Message Text" %>
```

```
<HTML>

Some text on the Web page.

</HTML>
```

In the next example, the page is buffered, and as a result, the server will not send output to the client until all the ASP scripts on the page have been processed or until the **Flush** method is called. With buffered output, calls to the **AddHeader** method can appear anywhere the script, so long as they precede any calls to the **Flush** method. If the call to the **AddHeader** method appeared below the call to the **Flush** method in the preceding example, the script would generate a run-time error.

```
<% Response.Buffer = TRUE %>

' Here is some text on your Web page.

<% Response.AddHeader "WARNING", "Error Message Text" %> Here's some
more interesting and illuminating text.

<% Response.Flush %>

<%= Response.Write("some string") %>
```

Examples: ASP Response Object AddHeader Method

The following example uses the **AddHeader** method to request that the client use BASIC authentication.

```
<% Response.Addheader "WWW-Authenticate", "BASIC" %>
```

Note

The preceding script merely informs the client browser which authentication to use. If you use this script in your Web applications, you should ensure that the Web server has BASIC authentication enabled.

ASP Response Object AppendToLog Method

The **AppendToLog** method adds a string to the end of the Web log entry for this page request. You can call it multiple times during the execution of a page; each time the string is appended to the existing entry.

Syntax: ASP Response Object AppendToLog Method

`Response.AppendToLog` *`string`*

Parameters: ASP Response Object AppendToLog Method

string

The text to append to the log. Because fields in Web server logs are often comma-delimited, this string cannot contain any commas. The maximum length of the string is 80 characters.

ASP Response Object BinaryWrite Method

The **BinaryWrite** method writes the specified information to the current HTTP output without any character conversion. It is useful for sending non-string information, such as binary data required by custom applications.

Syntax: ASP Response Object BinaryWrite Method

```
Response.BinaryWrite data
```

Parameters: ASP Response Object BinaryWrite Method

data

The binary information to be sent.

Examples: ASP Response Object BinaryWrite Method

If you have an object that creates an array of bytes, you can send the results using **BinaryWrite**:

```
<%
Set bg = Server.CreateObject(MY.BinaryGenerator)
Pict = bg.MakePicture
Response.BinaryWrite Pict
%>
```

ASP Response Object Clear Method

The **Clear** method erases any buffered HTML output. It only erases the response body, it does not affect headers. You can use this method to handle error messages. Calling **Clear** will cause an error if **Response.Buffer** is not TRUE.

Syntax: ASP Response Object Clear Method

```
Response.Clear
```

ASP Response Object End Method

The **End** method stops the Web server from processing any additional script and sends the current result. The remaining contents of the file are not processed.

Syntax: ASP Response Object End Method

```
Response.End
```

Remarks: ASP Response Object End Method

If **Response.Buffer** is set to TRUE, **End** flushes the buffer. If you do not want the result sent to the client, use the following:

```
Response.Clear
Response.End
```

ASP Response Object Flush Method

The **Flush** method sends buffered output immediately. **Flush** will cause a run-time error if called when **Response.Buffer** is not TRUE.

Syntax: ASP Response Object Flush Method

```
Response.Flush
```

Remarks: ASP Response Object Flush Method

If **Flush** is called on an ASP page, the server does not honor Keep-Alive requests for that page.

ASP Response Object Redirect Method

The **Redirect** method causes the browser to attempt to connect to a different URL.

Syntax: ASP Response Object Redirect Method

```
Reponse.Redirect URL
```

Parameters: ASP Response Object Redirect Method

URL

The Uniform Resource Locator the client is redirected to.

Remarks: ASP Response Object Redirect Method

Any response body content set explicitly in the page is ignored. However, the method does send to the client other HTTP headers set by this page. An automatic response body containing the redirect URL as a link is generated.

The **Redirect** method sends the following explicit header:

```
HTTP/1.0 302 Object Moved

Location URL
```

ASP Response Object Write Method

The **Write** method writes a specified string to the current output.

Syntax: ASP Response Object Write Method

```
Response.Write variant
```

Parameters: ASP Response Object Write Method

variant

The data to write. This parameter can be any data type supported by the Visual Basic VARIANT data type, including characters, strings, and integers. This value cannot contain the character

combination "%>"; instead you should use the escape sequence "%\>". The Web server will translate the escape sequence when it processes the script.

Remarks: ASP Response Object Write Method

VBScript limits the size of string literals to 1022 bytes, therefore *variant* cannot be a string literal of more than 1022 bytes. You can, however, specify *variant* as the name of a variable containing more than 1022 bytes.

Examples: ASP Response Object Write Method

The following VBScript, in which a is repeated 1023 times in the string literal will fail:

```
<% Response.Write "aaaaaaaaaaaa...aaaaaaaaaaaaaaaaaa"
```

The following VBScript, in which `a' is repeated 4096 times in the string variable will succeed:

```
AVeryLongString = String(4096, "a")

Response.Write(AVeryLongString)
```

Using the **Response.Write** method to send output to the client:

```
I just want to say <% Response.Write "Hello World." %>

Your name is: <% Response.Write Request.Form("name") %>
```

The following script demonstrates adding an HTML tag to the Web page output. Because the string returned by the **Write** method cannot contain the character combination, "%>", the escape, "%\>", has been used instead:

```
<% Response.Write "<TABLE WIDTH = 100%\>" %>
```

The script outputs:

```
<TABLE WIDTH = 100%>
```

ASP Response Object Properties

ASP Response Object Buffer Property

The **Buffer** property determines whether to buffer page output. When page output is buffered, HTML output is not sent to the client until the script on the page has been processed or until the ASP Response Object Flush or ASP Response Object End methods are called.

The **Buffer** property cannot be set after the server has sent output to the client. For this reason, you should set the **Buffer** property on the first line of the script.

Syntax: ASP Response Object Buffer Property

```
Response.Buffer = flag
```

Parameters: ASP Response Object Buffer Property

flag

Specifies whether to buffer page output. It can be one of the following values:

Value	Description
TRUE	Output is buffered. The server does not send output from the script on the page until all the script has been processes or until the **Flush** or **End** method is called.
FALSE	Output is not buffered. The server sends output from the script on the page as the script is processed.

Remarks: ASP Response Object Buffer Property

If the current ASP script has buffering set to TRUE and does not call the ASP Response Object Flush Method**,** the server will honor Keep-Alive requests made by the client. This saves time because the server does not have to create a new connection for each client request.

However, buffering prevents any of the response from being displayed to the client until the server has finished all script processing for the current page. For long scripts, this may cause a perceptible delay.

ASP Response Object CacheControl Property

The **CacheControl** property overrides the default **Private** value. Setting this property to **Public** allows proxy servers to cache the output generated by ASP.

Syntax: ASP Response Object CacheControl Property

```
Response.CacheControl = Cache Control Header
```

Parameters: ASP Response Object CacheControl Property

Cache Control Header

A cache control header that will be either Public or Private.

ASP Response Object Charset Property

The **Charset** property appends the name of the character set (for example, ISO-LATIN-7) to the content-type header in the response object.

Syntax: ASP Response Object Charset Property

```
Response.Charset(CharsetName)
```

Parameters: ASP Response Object Charset Property

```
CharsetName
```

A string that specifies a character set for the page. The character set name will be appended to the content-type header in the **Response** object.

Examples: ASP Response Object Charset Property

For an ASP page that did not include the **Response.Charset** property, the content-type header would be:

```
content-type:text/html
```

If the same .asp file included:

```
<% Response.Charset("ISO-LATIN-7") %>
```

the content-type header would be:

```
content-type:text/html; charset=ISO-LATIN-7
```

Remarks: ASP Response Object Charset Property

This function inserts any string in the header, whether it represents a valid character set or not.

If a single page contains multiple tags containing **Response.Charset**, each **Response.Charset** will replace the previous **CharsetName**. As a result, the character set will be set to the value specified by the last instance of **Response.Charset** in the page.

ASP Response Object ContentType Property

The **ContentType** property specifies the HTTP content type for the response. If not specified, the default is "text/HTML."

Syntax: ASP Response Object ContentType Property

```
Response.ContentType = ContentType
```

Parameters: ASP Response Object ContentType Property

ContentType

A string describing the content type. This string is usually formatted *type/subtype,* where *type* is the general content category and *subtype* is the specific content type. For a full list of supported content types, see your Web browser documentation or the current HTTP specification.

Examples: ASP Response Object ContentType Property

The following example sets the content type to non-HTML encoded text. This means the client will not interpret any HTML tags in the text:

```
<% Response.ContentType="text/plain"
```

The following example shows some other common content types:

```
<% Response.ContentType = "text/html" %>

<% Response.ContentType = "image/GIF" %>

<% Response.ContentType = "image/JPEG" %>
```

ASP Response Object Expires Property

The **Expires** property sets the length of time a page will be cached on a client browser. If the user returns to the page before it expires, the cached version will be displayed.

Syntax: ASP Response Object Expires Property

```
Response.Expires = number
```

Parameters: ASP Response Object Expires Property

number

The number of minutes until the page expires. Set this to 0 to have the cached page expire immediately.

Remarks: ASP Response Object Expires Property

If the property is set more than once on a page, the shortest time is used.

ASP Response Object ExpiresAbsolute Property

The **ExpiresAbsolute** property specifies the date and time at which a page cached on a browser expires. If the user returns to the same page before that date and time, the cached version is displayed. If a time is not specified, the page expires at midnight of that day. If a date is not specified, the page expires at the given time on the day that the script is run.

Syntax: ASP Response Object ExpiresAbsolute Property

```
Reponse.ExpiresAbsolute = [date] [time]
```

Parameters: ASP Response Object ExpiresAbsolute Property

date

Specifies the date on which the page will expire. The value sent in the Expires header conforms to the RFC-1123 date format.

time

Specifies the time at which the page will expire. This value is converted to GMT before the header is sent.

Remarks: ASP Response Object ExpiresAbsolute Property

If this property is set more than once on a page, the earliest time is used.

Examples: ASP Response Object ExpiresAbsolute Property

The following example sets the page to expire 15 seconds after 1:30 p.m. on May 31, 1996:

```
Response.ExpiresAbsolute = #May 31, 1996 13:30:15#
```

ASP Response Object IsClientConnected Property

The **IsClientConnected** property is a read-only property that indicates if the client has disconnected since the last call to **Response.Write**.

Syntax: ASP Response Object IsClientConnected Property

```
Response.IsClientConnected
```

Remarks: ASP Response Object IsClientConnected Property

This property allows you greater control over circumstances where the client may have disconnected from the server. For example, if a long period of time has elapsed between a client request and the server response, it may be beneficial to make sure the client is still connected before continuing to process the script.

Examples: ASP Response Object IsClientConnected Property

```
<%
'check to see if the client is connected
If Not Response.IsClientConnected Then
  'get the sessionid to send to the shutdown function
  Shutdownid = Session.SessionID
'perform shutdown processing
  Shutdown(Shutdownid)
  End If
%>
```

ASP Response Object PICS Property

The PICS property adds a value to the pics-label field of the response header.

Syntax: ASP Response Object PICS Property

```
Response.PICS(PICSLabel)
```

Parameters: ASP Response Object PICS Property

PICSLabel

A string that is a properly formatted PICS label. The value will be appended to the PICS-Label field in the response header.

Remarks: ASP Response Object PICS Property

The **Response.PICS** property inserts any string into the response header, whether or not it is a valid PICS label.

If a single page sets **Response.PICS** multiple times, each setting will replace the previous one. As a result, the PICS label will be set to the last **Response.PICS** instance on the page.

Because PICS labels contain quotes, you must replace quotes with `" & chr(34) & ".`

Examples: ASP Response Object PICS Property

For an .asp file that includes:

```
<%
Response.PICS("(PICS-1.1 <http://www.rsac.org/ratingv01.html>
```

```
labels on " & chr(34) & "1997.01.05T08:15-0500" & chr(34) &
" until" & chr(34) & "1999.12.31T23:59-0000" & chr(34) &
" ratings (v 0 s 0 l 0 n 0))")
%>
```

the following header would be added:

```
PICS-label:(PICS-1.1 <http://www.rsac.org/ratingv01.html>
labels on "1997.01.05T08:15-0500"
until "1999.12.31T23:59-0000"
ratings (v 0 s 0 l 0 n 0))
```

ASP Response Object Status Property

The **Status** property sets the value of the status line returned by the server. Status values are defined in the HTTP specification.

Syntax: ASP Response Object Status Property

`Response.Status` = *`StatusDescription`*

Parameters: ASP Response Object Status Property

StatusDescription

A string that contains a three-digit status code and a brief explanation of that status. For example, `"310 Move Permanently".`

Remarks: ASP Response Object Status Property

Use this property to modify the status line returned by the server.

Examples: ASP Response Object Status Property

The following example sets the response status:

```
Response.Status = "401 Unauthorized"
```

ASP Server Object

The **Server** object provides access to methods and properties on the server. Most of its methods and properties serve as utility functions.

Syntax: ASP Server Object

`Server.method` | *`property`*

ASP Server Object Properties

ASP Server Object ScriptTimeout Property	The length of time a script runs before it is terminated.

ASP Server Object Methods

ASP Server Object CreateObject Method	Creates an instance of a server component.
ASP Server Object HTMLEncode Method	Applies HTML encoding to a specified string.
ASP Server Object MapPath Method	Maps the specified virtual path, either the absolute path on the current server or the path relative to the current page, into a physical path.
ASP Server Object URLEncode Method	Applies URL encoding rules, including escape characters, to a string.

ASP Server Object Methods

ASP Server Object CreateObject Method

The **CreateObject** method creates an instance of a server component. If the component has implemented the **OnStartPage** and **OnEndPage** methods, the **OnStartPage** method is called at this time.

Syntax: ASP Server Object CreateObject Method

Obj = **Server.CreateObject(** *progID* **)**

Parameters: ASP Server Object CreateObject Method

Obj

A variable name for the object

progID

Specifies the type of object to create. The format for *progID* is [*vendor.*]*component*[*.version*].

Remarks: ASP Server Object CreateObject Method

By default, objects created by the **Server.CreateObject** method have page scope. This means that they are automatically destroyed by the server when it finishes processing the current ASP page.

To create an object with session or application scope, you can either use the <OBJECT> tag and set the SCOPE parameter to SESSION or APPLICATION, or store the object in a session or application variable.

Examples: ASP Server Object CreateObject Method

You can destroy an object by setting the variable to NOTHING or setting the variable to a new value, as shown below. The first example releases the object ad. The second replaces ad with a string:

```
<% Session("ad") = Nothing %>

<% Session("ad") = "some other value" %>
```

You cannot create an object with the same name as a built-in object. The following example will cause an error:

```
<% Set Response = Server.CreateObject("Response") %>
```

The following example creates an instance of the **MSWC.BrowserType** component. This component can be used to determine the capabilities of the browser requesting the page.

```
<% Set MyB = Server.CreateObject("MSWC.BrowserType") %>
```

ASP Server Object HTMLEncode Method

The **HTMLEncode** method applies HTML encoding to a specified string.

Syntax: ASP Server Object HTMLEncode Method

Server.HTMLEncode(*string* **)**

Parameters: ASP Server Object HTMLEncode Method

string

Specifies the string to encode.

Examples: ASP Server Object HTMLEncode Method

The following script:

```
<% Server.HTMLEncode("The paragraph tag <P>" %>
```

produces the following output:

```
The paragraph tag &lt;P&gt;
```

The preceding text will be displayed on a Web browser as:

```
The paragraph tag <P>
```

You can view the source to see the encoded HTML.

ASP Server Object MapPath Method

The **MapPath** method maps the specified relative or virtual path to the corresponding physical directory on the server.

Syntax: ASP Server Object MapPath Method

Server.MapPath(*path* **)**

Parameters: ASP Server Object MapPath Method

path

Specifies the relative or virtual path to map to a physical directory. If *path* starts with either a forward or backward slash, either (/) or (\), the **MapPath** method returns a path as if *path* is a full virtual path. If *path* doesn't start with a slash, the **MapPath** method returns a path relative to the directory of the .ASP file being processed.

Note

The path parameter can contain relative paths (`../../Scripts/`, for example).

Remarks: ASP Server Object MapPath Method

The **MapPath** method does not check whether the path it returns is valid or exists on the server. Because the **MapPath** method maps a path regardless of whether the specified directories currently exist, you can use the **MapPath** method to map a path to a physical directory structure, and then pass that path to a component that creates the specified directory or file on the server.

Examples: ASP Server Object MapPath Method

For the examples below, the DATA.TXT and TEST.ASP files are located in the C:\Inetpub\Wwwroot\Script directory. The TEST.ASP file contains scripts. The C:\Inetpub\Wwwroot directory is set as the server's home directory.

The following example uses the server variable PATH_INFO to map the physical path to the current file:

```
<%= server.mappath(Request.ServerVariables("PATH_INFO"))%>
<BR>
```

This script produces the following:

```
c:\inetpub\wwwroot\script\test.asp<BR>
```

Because the path parameters in the following examples do not start with a slash character, they are mapped relative to the current directory, in this case C:\Inetpub\Wwwroot\Script.

```
<%= server.mappath("data.txt")%><BR>
<%= server.mappath("script/data.txt")%><BR>
```

This script outputs the following:

```
c:\inetpub\wwwroot\script\data.txt<BR>
c:\inetpub\wwwroot\script\script\data.txt<BR>
```

The next two scripts use the slash characters to specify that the paths returned should be looked up as complete virtual paths on the server:

```
<%= server.mappath("/script/data.txt")%><BR>
<%= server.mappath("\script")%><BR>
```

This script outputs the following:

```
c:\inetpub\script\data.txt<BR>

c:\inetpub\script<BR>
```

The following examples demonstrate how you can use either a forward slash (/) or a backslash (\) to return the physical path to the home directory. The following script:

```
<%= server.mappath("/")%><BR>

<%= server.mappath("\")%><BR>
```

produces the following output:

```
c:\inetpub\wwwroot<BR>

c:\inetpub\wwwroot<BR>
```

ASP Server Object URLEncode Method

The **URLEncode** method applies URL encoding rules, including escape characters, to a specified string.

Syntax: ASP Server Object URLEncode Method

Server.URLEncode(*string* **)**

Parameters: ASP Server Object URLEncode Method

string

Specifies the string to encode.

Examples: ASP Server Object URLEncode Method

The following example encodes the paragraph tag:

```
<%= Server.URLEncode("The paragraph tag: <P>") %>
```

The script produces the following:

```
The+paragraph+tag%3A+%3CP%3E
```

ASP Server Object Properties

ASP Server Object ScriptTimeout Property

The **ScriptTimeout** property specifies the maximum amount of time a script can run before it is terminated. The delay before scripts are ended is by default 90 seconds. This will not take effect while a server component is processing.

Syntax: ASP Server Object ScriptTimeout Property

Server.ScriptTimeout = *NumSeconds*

Parameters: ASP Server Object ScriptTimeout Property

NumSeconds

Specifies the maximum number of seconds that a script can run before the server terminates it. The default value is 90 seconds.

Note

The **ScriptTimeout** property cannot be set to a value less than that specified in the registry settings or configuration file. For example, if *NumSeconds* is set to 10, and the registry setting or configuration file contains the default value of 90 seconds, scripts will time out after 90 seconds. However, if *NumSeconds* were set to 100, the scripts would time out after 100 seconds.

Examples: ASP Server Object ScriptTimeout Property

The following example causes scripts to time out if the server takes longer than 30 seconds to process them.

```
<% Server.ScriptTimeout = 30 %>
```

The following example retrieves the current value of the **ScriptTimeout** property and stores it in the variable *TimeOut*.

```
<% TimeOut = Server.ScriptTimeout %>
```

ASP Session Object

The **Session** object stores information needed for a particular user-session.

Variables stored in the **Session** object are not discarded when the user jumps between pages in the application; instead, they persist for the entire user-session. The Web server automatically creates a **Session** object when a Web page (from a server application) is requested by a user who does not already have a session. The server destroys the **Session** object when the session expires or is abandoned.

The **AllowSessionState** registry or configuration file setting controls the creation of **Session** objects. If **AllowSessionState** is set to **False**, the **Session** object cannot be used. The default is to use Sessions.

Session object event scripts are declared in the Global.asa.

Note

Session state is only maintained for browsers that support cookies.

Syntax: ASP Session Object

Session.*collection*|*property*|*method*

ASP Session Object Collections

ASP Session Object Contents Collection	Contains the items you have added to the session with script commands.

ASP Session Object StaticObjects Collection	Contains items created in Global.asa using the <OBJECT> tag and given session scope.

ASP Session Object Properties

ASP Session Object SessionID Property	Returns the session identifier for this client. Each session has a unique identifier.
ASP Session Object Timeout Property	The timeout period, in minutes, for session state for this application.
ASP Session Object LCID Property	Sets or gets a Locale Identifier (LCID) that determines how certain content—such as date, time, and currency—is formatted.

The **CodePage** property is not currently implemented in Sun Chili!Soft ASP.

ASP Session Object Methods

ASP Session Object Abandon Method	Destroys a **Session** object and all objects stored in it, and releases their resources.

ASP Session Object Events

Session_OnStart	Occurs when the server creates a new session. It runs before executing the requested page.
Session_OnEnd	Occurs when the session is abandoned or times out.

Session object events are defined in the Global.asa file.

Remarks: ASP Session Object

You can store values in the **Session** object. Information stored in the **Session** object is available for the entire session and has session scope. The following script demonstrates how two types of variables are stored:

```
Session("username") = "Janine"

Session("age") = 42
```

If you are using VBScript as your scripting language, you must use the **Set** keyword to store an object in the **Session** object, as shown in the following example:

```
<% Set Session("Obj1") = Server.CreateObject("MyComponent") %>
```

You can then call the methods and properties of *Obj1* on subsequent Web pages by using the following syntax:

```
<% Session("Obj1").MyObjMethod %>
```

As an alternative, you can extract a local copy of the object:

```
Set MyLocalObj = Session("Obj1")

MyLocalObj.MyObjMethod
```

You cannot store a built-in object in a **Session** object. Each of the following lines will return an error:

```
Set Session("var1") = Session

Set Session("var2") = Request

Set Session("var3") = Response

Set Session("var4") = Server

Set Session("var5") = Application
```

Before you store an object in the **Session** object, you must know what threading model it uses. Only objects marked as both free and apartment-threaded can be stored in the **Session** object.

The **Session** object is implemented as a collection. If you store an array in an **Session** object, you should not attempt to alter elements of the stored array directly. For example, the following script does not work:

```
Session("StoredArray") (3) = "new value"
```

Instead of storing the value "new value" in `StoredArray(3)`, the value is stored in the **Session** collection, overwriting any information stored at `Session(3)`.

See the **Application** object for an example of storing an array.

ASP Session Object Collections

ASP Session Object Contents Collection

The **Contents** collection contains all of the items that have been created for a session without using the `<OBJECT>` tag. The collection can be used to determine the value of a specific item, or to iterate over all the items in the session.

Syntax: ASP Session Object Contents Collection

Session.Contents(*key* **)**

Parameters: ASP Session Object Contents Collection

key

The name of the item to retrieve.

Remarks: ASP Session Object Contents Collection

You can use an iterating control structure to loop through the keys of the **Contents** collection. This is demonstrated in the following example.

```
<%

Dim sessitem

For Each sessitem in Session.Contents
```

```
  Response.write(sessitem & " : " & Session.Contents(sessitem) &
"<BR>")
Next
%>
```

ASP Session Object StaticObjects Collection

The **StaticObjects** collection contains all the objects created in Global.asa with the <OBJECT> tag and given session scope. The collection can be used to retrieve the value of a specific item, or you can use an iterator to retrieve all the items in the collection.

Syntax: ASP Session Object StaticObjects Collection

Session.StaticObjects(*key* **)**

Parameters: ASP Session Object StaticObjects Collection

key

The item to retrieve.

Remarks: ASP Session Object StaticObjects Collection

You can use an iterating control structure to loop through the keys of the **StaticObjects** collection. This is demonstrated in the following example.

```
<%
Dim objprop
For Each objprop in Session.StaticObjects
  Response.write(objproperty & " : " &
Session.StaticObjects(objprop) & "<BR>")
Next
%>
```

ASP Session Object Methods

ASP Session Object Abandon Method

The **Abandon** method destroys all the objects stored in a **Session** object and releases their resources. If you do not call the **Abandon** method explicitly, the server destroys these objects when the session times out.

Syntax: ASP Session Object Abandon Method

Session.Abandon

Remarks: ASP Session Object Abandon Method

When the **Abandon** method is called, the current **Session** object is queued for deletion, but is not actually deleted until all of the script commands on the current page have been processed. This means that you can access variables stored in the **Session** object on the same page as the call to **Abandon**, but not in any subsequent Web pages.

For example, in the following script, the third line prints the value Mary. This is because the **Session** object is not destroyed until the server has finished processing the script.

```
Session.Abandon

Session("MyName") = "Mary"

Response.Write(Session("MyName"))
```

If you access the variable `MyName` on a subsequent Web page, it is empty. This is because `MyName` was destroyed with the previous **Session** object when the page containing the above example finished processing.

The server creates a new **Session** object when you open a subsequent Web page after abandoning a session. You can store variables and objects in this new **Session** object.

ASP Session Object Properties

ASP Session Object SessionID Property

The **SessionID** property returns the session identification for this user. Each session has a unique identifier that is generated by the server when the session is created. The session ID is returned as data type **LONG.**

Syntax: ASP Session Object SessionID Property

Session.SessionID

Remarks: ASP Session Object SessionID Property

Do not use the **SessionID** property to generate primary key values for a database application. This is because if the Web server is restarted, some **SessionID** values may be the same as those generated before the server was stopped. Instead, you should use an auto-increment column data type.

ASP Session Object Timeout Property

The **Timeout** property specifies the timeout period for the **Session** object for this application, in minutes. If the user does not refresh or request a page within the timeout period, the session ends.

Syntax: ASP Session Object Timeout Property

Session.Timeout [= *nMinutes*]

Parameters: ASP Session Object Timeout Property

nMinutes

The number of minutes that a session can remain idle before the server terminates it automatically. The default is 20 minutes.

Remarks: ASP Session Object Timeout Property

The **SessionTimeout** registry or configuration file setting controls the default value of the **Timeout** property for an ASP application.

ASP Session Object LCID Property

The **SessionID** property enables you to set or get a Locale Identifier (LCID) that determines how certain content—such as date, time, and currency—is formatted.

Syntax: ASP Session Object LCID Property

nCurrentLCID = **Session.LCID**

SessionLCID = *nLCIDnumber*

Parameters: ASP Session Object LCID Property

nLCIDnumber

A valid Local Identifier (LCID) number. For a list of valid values, see "Developing International Applications" in "Chapter 4: Building a Sun Chili!Soft ASP Application."

Remarks: ASP Session Object LCID Property

For usage and limitations, see "Developing International Applications" in "Chapter 4: Building a Sun Chili!Soft ASP Application."

ASP Component Reference

Sun Chili!Soft ASP automatically installs a number of components that you can use to build dynamic Web pages. This section provides reference information about these components.

In this section:

- Installed ASP Components
- ASP Ad Rotator Component
- ASP Browser Capabilities Component
- ASP Content Linking Component
- ASP Content Rotator Component
- ASP Counters Component
- ASP MyInfo Component
- ASP Tools Component

Installed ASP Components

The following is a list of ASP components that are installed with Sun Chili!Soft ASP.

Component	Description
Ad Rotator	Creates an **Ad Rotator** object that automates the rotation of advertisement images on a Web page.
Browser Capabilities	Creates a **BrowserType** object that determines the type, version, and capabilities of every browser that visits your site.
Content Linking	Creates a **NextLink** object that manages a list of URLs so that you can treat the pages in your Web site like the pages in a book.
Content Rotator	Creates a **ContentRotator** object that automatically rotates HTML content strings on a Web page.
Counters	Creates a **Counters** object that can create, store, increment, and retrieve any number of individual counters.
MyInfo	Creates a **MyInfo** object that keeps track of personal information, such as the site administrator's name, address, and display choices.
Tools	Creates a **Tools** object that provides utilities that enable you to easily add sophisticated functionality to your Web pages.

ASP Ad Rotator Component

The Ad Rotator component creates an **Ad Rotator** object that automates the rotation of advertisement images on a Web page. Each time a user opens or reloads the Web page, the **Ad Rotator** object displays a new advertisement based on the information you specify in a Rotator Schedule file.

You can record how many users click each advertisement by setting the URL parameter in the Rotator Schedule file to direct users to the Redirection file. When you specify this parameter, each jump to an advertiser’s URL is recorded in the Web server activity logs.

The Ad Rotator object relies on two additional files for parameters and functionality:

Redirection File. An optional file that implements redirection and enables you to record how many users click on each advertisement and save this information to a file on the server.

Rotator Schedule File. A text file that contains the display schedule and file information for advertisements. This file must be available on a Web server virtual path.

Registry Settings: ASP Ad Rotator Component

The Ad Rotator Component makes use of no registry settings.

Syntax: ASP Ad Rotator Component

The Ad Rotator Control is registered with the ProgId of "MSWC.AdRotator". The following VBScript excerpt shows creating an instance of the control.

```
Set adRot = Server.CreateObject( "MSWC.AdRotator" )
```

Properties: ASP Ad Rotator Component

- **Border**
- **Clickable**
- **TargetFrame**

ASP Ad Rotator Component Rotator Schedule File

The Rotator Schedule file contains information that the Ad Rotator component uses to manage and display the various advertisement images. In it you can specify the details for the advertisements, such as the size of the advertisement space, the image files to use, and the percentage of time that each file should be displayed.

The Rotator Schedule file has two sections. The first section sets parameters that apply to all advertisement images in the rotation schedule. The second section specifies file and location information for each individual advertisement and the percentage of display time that each advertisement should receive. The two sections are separated by a line containing only an asterisk (*).

In the first section there are four global parameters, each consisting of a keyword and a value. All are optional. If you do not specify values for the global parameters, the Ad Rotator uses default values. In this case, the first line of the file must contain only an asterisk (*).

Syntax: ASP Ad Rotator Component Rotator Schedule File

[**REDIRECT** *URL*]

[**WIDTH** *numWidth*]

[**HEIGHT** *numHeight*]

[**BORDER** *numBorder*]

*

adURL

adHomePageURL

Text

impressions

Parameters: ASP Ad Rotator Component Rotator Schedule File

URL

Specifies the path to the dynamic-link library (.dll) or application (.asp) file that implements redirection. This path can be specified either fully (http://MyServer/MyDir/redirect.asp) or relative to the virtual directory (/MyDir/redirect.asp).

numWidth

Specifies the width of the advertisement on the page, in pixels. The default is 440 pixels.

numHeight

Specifies the height of the advertisement on the page, in pixels. The default is 60 pixels.

numBorder

Specifies the thickness of the hyperlink border around the advertisement, in pixels. The default is a 1-pixel border. Set this parameter to 0 for no border.

adURL

The location of the advertisement image file.

adHomePageURL

The location of the advertiser's home page. If the advertiser does not have a home page, put a hyphen (-) on this line to indicate that there is no link for this ad.

Text

Alternate text that is displayed if the browser does not support graphics, or has its graphics capabilities turned off.

impressions

A number between 0 and 10000 that indicates the relative weight of the advertisement.

For example, if a Rotator Schedule file contains three ads with impressions set to 2, 3, and 5, the first advertisement is displayed 20 percent of the time, the second 30 percent of the time, and the third 50 percent of the time.

Remarks: ASP Ad Rotator Component Rotator Schedule File

If the sum of the *impressions* parameters for all items exceeds 10000, an error will be generated the first time the Rotator Schedule file is accessed by a call to the **GetAdvertisement** method.

Examples: ASP Ad Rotator Component Rotator Schedule File

The following script demonstrates how you can use a rotator schedule file to display a variety of advertisements and how to include a redirection file.

```
---ADROT.TXT---

REDIRECT /scripts/adredir.asp

WIDTH 440

HEIGHT 60

BORDER 1

*
```

```
http://kabaweb/ads/homepage/chlogolg.gif
http://www.bytecomp.com/
Check out the ByteComp Technology Center
20
http://kabaweb/ads/homepage/gamichlg.gif
-
Sponsored by Flyteworks
20
http://kabaweb/ads/homepage/ismodemlg.gif
http:// www.proelectron.com/
28.8 internal PC modem, only $99
80
http://kabaweb/ads/homepage/spranklg.gif
http://www.clocktower.com/
The #1 Sports site on the net
10
```

ASP Ad Rotator Component Redirection File

The Redirection file is a file that you create. It usually includes script to parse the query string sent by the **AdRotator** object and to redirect the user to the URL associated with the advertisement that the user clicked on.

You can also include script in the Redirection file to count the number of users that have clicked on a particular advertisement, and save this information to a file on the server.

Examples: ASP Ad Rotator Component Redirection File

The following example redirects the user to the advertiser's home page.

```
---ADREDIR.ASP---
<% Response.Redirect(Request.QueryString("url")) %>
```

ASP Ad Rotator Component Properties

ASP Ad Rotator Component Border Property

The **Border** property enables you to specify whether to display the advertisements with a surrounding border.

Syntax: ASP Ad Rotator Component Border Property

```
Border = size
```

Parameters: ASP Ad Rotator Component Border Property

size

Specifies the thickness of the border that surrounds the displayed advertisement. The default is the value set in the header of Rotator Schedule file. 0 specifies no border.

ASP Ad Rotator Component Clickable Property

The **Clickable** property enables you to specify whether the advertisements are displayed as hyperlinks.

Syntax: ASP Ad Rotator Component Clickable Property

Clickable = *value*

Parameters: ASP Ad Rotator Component Clickable Property

value

Specifies whether the advertisement should be a hyperlink. This parameter can set to either TRUE or FALSE. If FALSE, only the image is displayed without a "click-through" hyperlink. The default value is TRUE.

ASP Ad Rotator Component TargetFrame Property

The **TargetFrame** property specifies the target frame into which the link should be loaded. This property fulfills the same function as the TARGET parameter in an HTML anchor statement.

Syntax: ASP Ad Rotator Component TargetFrame Property

TargetFrame = *frame*

Parameters: ASP Ad Rotator Component TargetFrame Property

frame

Specifies the name of the frame in which to display the advertisement. This parameter can also be one of the HTML frame-keywords, such as _TOP, NEW, CHILD, _SELF, _PARENT, or _BLANK. The default value is NO FRAME.

ASP Ad Rotator Component Methods

ASP Ad Rotator Component GetAdvertisement Method

The **GetAdvertisement** method retrieves the next advertisement from the Rotator Schedule file. Each time the script is run, such as when a user opens or refreshes a page, the method retrieves the next scheduled advertisement.

Arguments: ASP Ad Rotator Component GetAdvertisement Method

rotationSchedulePath	Specifies the location of the Rotator Schedule file relative to the virtual directory. For example, if the physical path was C:\Inetpub\Wwwroot\Ads\Adrot.txt (where Wwwroot is the "/" virtual directory), you would specify the path \Ads\Adrot.txt.

Return Values: ASP Ad Rotator Component GetAdvertisement Method

Returns HTML that displays the advertisement in the current page.

Examples: ASP Ad Rotator Component GetAdvertisement Method

The following example gets an advertisement from the Adrot.txt file in the /Ads/ virtual directory.

```
<% Set NextAd = Server.CreateObject("MSWC.AdRotator") %>

<%= NextAd.GetAdvertisement("/ads/adrot.txt") %>
```

Examples: ASP Ad Rotator Component GetAdvertisement Method HTML Output

Assuming the following fragment of a redirection file is chosen by the control:

```
REDIRECT /foo/bar.asp

WIDTH 300

HEIGHT 40

BORDER 1

*

/ads/picture.gif

http://www.chilisoft.com/info/index.html

Hello from Chilisoft.

90
```

The HTML that is produced is:

```
<A HREF=
"/foo/bar.asp?url=http://www.chilisoft.com/info/index.html&image=/ad
s/picture.gif TARGET="_blank">

<IMG SRC="/ads/picture.gif" ALT="Hello from Chilisoft" WIDTH=300
HEIGHT=40 BORDER = 1>

</A>
```

The Redirect script "foo/bar.asp" is invoked and can record click-through information before redirecting the client browser to the user's desired location.

ASP Browser Capabilities Component

The Browser Capabilities component determines which features a browser supports. This component uses two files: Browscap.ini and Browscap.dll (libchilicap.so and libchilicap.ini on UNIX).

Syntax: ASP Browser Capabilities Component

```
Set BrowserType = Server.CreateObject("MSWC.BrowserType")
```

Parameters: ASP Browser Capabilities Component

BrowserType

Specifies the name of the object created by the call to **Server.CreateObject**.

When a client requests a page from the server, the HTTP header includes a **user agent** ASCII string that specifies the browser software name and version. The Browser Capabilities component searches for this string in the Browser Capabilities component Browsecap.ini file. When it finds a match, the properties of the client browser are read and the server adopts the properties of the browser.

The following table lists the minimum set of properties that ASP always checks:

Property	Description
ActiveXControls	Support for Active X Controls.
Backgroundsounds	Support for background sounds.
Beta	Is browser beta software?
Browser	Browser name.
Cookies	Support for cookies.
Frames	Support for frames.
Javaapplets	Support for Java applets.
Javascript	Support for JavaScript.
Majorver	Major version number of the browser.
Minorver	Minor version number of the browser.
Parent	Parent browser (as defined in `browscap.ini`).
Platform	User's operating system.
Tables	Support for HTML tables.
Vbscript	Support for VBScript.
Version	Full version number of the browser.

Browsecap.ini File: ASP Browser Capabilities Component

The Browscap.ini (called libchilicap.ini on UNIX) file contains information about each known browser. It is a standard text file that lists features a browser supports. The Browscap.ini file maps browser capabilities to the HTTP User Agent header.

It is important to keep your Browscap.ini or libchilicap.ini file up to date. When new browsers are released their capabilities are unknown to the current file, and pages that rely on browser detection may fail. You can obtain updates to Browscap.ini at:

http://www.cyscape.com/browscap/

To use the Browscap.ini file on UNIX, you must convert the text file to UNIX format and rename it "libchilicap.ini." You should rename your existing libchilicap.ini file "libchilicap.old" before installing the updated version.

You can also maintain the Browscap.ini file by editing the Browscap.ini file. A default section of the Browscap.ini file is used when the browser details don't match any of the ones specified. If the browser in use doesn't match any in the Browscap.ini file, and no default browser settings are specified, all properties are set "`UNKNOWN`."

Note

The Browscap.ini (or libchilicap.ini) file must be in the same directory as Browscap.dll or libchilicap.so.

To use the Browser Capabilities Component, it is necessary to create an instance of it and refer to its properties. To avoid having the Browscap.ini file accessed every time, read the value once and assign it to a variable:

```
Set objBCap = Server.CreateObject("MSWC.BrowserType")
```

Syntax: Browsecap.ini File HTTPUserAgentHeader Section

The `HTTPUserAgentHeader` section of Browscap.ini (libchilicap.ini on UNIX) defines the properties for a particular browser. The syntax is as follows:

```
[HTTPUserAgentHeader]
parent = browserDefinition
property1 = value 1
property2 = value 2
.
.
.
```

Parent

Another definition contains more information for that browser

value 1

A number used to map a capability for the first property listed.

value 2

A number used to map a capability for the second property listed.

Browsecap.ini File Default Section

The `Default` section of Browscap.ini (libchilicap.ini on UNIX) lists the properties and values to be used if the current browser isn't listed in its own section (or, if listed, not all properties are supplied). The following is the syntax for the default section of the Browscap.ini file:

```
[Default Browser Capability Settings]
defaultProperty1 = default value 1
defaultProperty2 = default value 2
.
.
.
```

default value 1

A number used to map a default capability for the first property listed.

default value 2

A number used to map a default capability for the second property listed.

Examples: Browsecap.ini File Default Section[0]

This example shows entries for Internet Explorer (IE) 3.0. Since it has no parent line, the only properties its has (other than those defined in the default section) are those explicitly defined:

```
[IE 3.0]
browser=IE
Version =3.0
majorver=#3
minorver=#0
frames=TRUE
tables=TRUE
cookies=TRUE
vbscript=TRUE
javascript=TRUE
ActiveXControls=TRUE
```

In the following example IE 3.0 is specified as the parent for the browser. The properties explicitly provided replace, or add to, those values in the parent's definition:

```
[Mozilla/2.0 (compatible; MSIE 3.01; Windows 95)]
parent=IE 3.0
version = 3.01
minorver=01
```

```
platform=Win95
[Default Browser Capability Settings]
browser=Default
frames=FALSE
tables=FALSE
cookies=FALSE
backgroundsounds=FALSE
vbscript= FALSE
javascript= FALSE
. . .
```

To determine if a browser supports JavaScript use the following code:

```
bc = Server.CreateObject("MSWC.BrowserType")
if bc.javascript = 0 then
Response.Write "This browser does not support JavaScript."
else
REM The browser supports JavaScript so simply continue.
end if
```

The following example determines if a browser supports tables:

```
bc = Server.CreateObject("MSWC.BrowserType")
if bc.tables = 0 then
Response.Write "This browser does not support tables."
. . .
```

The following example uses the Browser Capabilities component to display a table showing some of the capabilities of the current browser:

```
<% Set bc = Server.CreateObject("MSWC.BrowserType") %>
<table border=1>
<tr><td>Browser</td><td> <%= bc.browser %>
<tr><td>Version</td><td> <%= bc.version %> </td></TR>
<tr><td>Frames</td><td>
<% if (bc.frames = TRUE) then %> TRUE
<% else %> FALSE
<% end if %> </td></TR>
<tr><td>Tables</td><td>
```

```
<% if (bc.tables = TRUE) then %> TRUE
<% else %> FALSE
<% end if %> </td></TR>
<tr><td>BackgroundSounds</td><td>
<% if (bc.BackgroundSounds = TRUE) then %> TRUE
<% else %> FALSE
<% end if %> </td></TR>
<tr><td>VBScript</td><td>
<% if (bc.vbscript = TRUE) then %> TRUE
<% else %> FALSE
<% end if %> </td></TR>
<tr><td>JScript</td><td>
<% if (bc.javascript = TRUE) then %> TRUE
<% else %> FALSE
<% end if %> </td></TR>
</table>
```

ASP Content Linking Component

The Content Linking component creates a **NextLink** object that manages a list of URLs so that you can treat the pages in your Web site like the pages in a book. You can use the Content Linking component to automatically generate and update tables of contents and navigational links to previous and subsequent Web pages. This is ideal for applications such as online newspapers and forum message listings.

The Content Linking component references a Content Linking List file that contains the list of the linked Web pages. This list is stored on the Web server and must be available on a web server virtual path.

Registry Settings: ASP Content Linking Component

The control makes use of no registry settings.

Syntax: ASP Content Linking Component

The Content Linking component is registered with the ProgId of "MSWC.NextLink." The following VBScript excerpt shows creating an instance of the control.

```
Set cLinker = Server.CreateObject( "MSWC.NextLink" )
```

Properties: ASP Content Linking Component

None.

Methods: ASP Content Linking Component

- **GetListCount**
- **etListIndex**
- **GetNextDescription**
- **GetNextURL**
- **GetNthDescription**
- **GetNthURL**
- **GetPreviousDescription**
- **GetPreviousURL**

Examples: ASP Content Linking Component

The following example builds a table of contents.

```
<OL>
<%
 Set NextLink = Server.CreateObject ("MSWC.NextLink")
 count = NextLink.GetListCount ("/data/nextlink.txt")
 I = 1
%>
<UL>
<% Do While (I <= count) %>
<LI><A HREF=" <%= NextLink.GetNthURL ("/data/nextlink.txt", I) %> ">
<%= NextLink.GetNthDescription ("/data/nextlink.txt", I) %> </A>
<%
 I = (I + 1)
 Loop
%>
</UL>
</OL>
```

The following script adds the next-page and previous-page buttons to an HTML file.

```
<%
```

```
 Set NextLink = Server.CreateObject ("MSWC.NextLink")
 If (NextLink.GetListIndex ("/data/nextlink.txt") > 1)
 Then
%>
<A HREF=" <%= NextLink.GetPreviousURL ("/data/nextlink.txt") %> ">
Previous Page</A>
<% End If %>
<A HREF=" <%= NextLink.GetNextURL ("/data/nextlink.txt") %> ">Next
Page</A>
```

ASP Content Linking Component Content Linking List File

The Content Linking List file contains one line of text for each URL in the list. Each line ends in a carriage return and each item on a line is separated by a TAB character.

Syntax: ASP Content Linking Component Content Linking List File

Web-page-URL [*text-description* [*comment*]]

Values[0]: ASP Content Linking Component Content Linking List File

Web-page-URL

The virtual or relative URL of the Web page in the format *filename* or *directory\filename.* Absolute URLs, those that start with "http:", "//", or "\\", are not supported and will not be processed by methods such as **GetNextURL** and **GetListIndex**. When building your content path, you should ensure that no collisions or infinite loops can occur.

text-description

A value containing text that describes *Web-page-URL.*

comment

Explanatory text that is not processed by the component.

Examples: ASP Content Linking Component Content Linking List File

The following text file creates a list of URLs that can be used by the Content Linking component.

```
---NEXTLINK.TXT---
story1.htm Highlights From the Hockey Playoffs
story2.htm Congress Passes New Welfare Initiative
story3.htm Cider Recipes to Warm Long Winter Nights
story4.htm Winter Storm to bring more snow to East
story5.htm Reducing Stress on the Job
```

```
main.htm Return to the table of contents
```

ASP Content Linking Component Methods

ASP Content Linking Component GetListCount Method

The **GetListCount** method retrieves the total number of Web pages listed in the Content Linking List file.

Arguments: ASP Content Linking Component GetListCount Method

listURL	The location of the Content Linking List file.

Return Values: ASP Content Linking Component GetListCount Method

This method returns an integer.

ASP Content Linking Component GetListIndex Method

The **GetListIndex** method retrieves the index number of the current item in the Content Linking List file.

Arguments: ASP Content Linking Component GetListIndex Method

listURL	The location of the Content Linking List file.

Return Values: ASP Content Linking Component GetListIndex Method

The **GetListIndex** method returns an integer index value specifying the current page's position on the file list. The index number of the first item is 1. The method returns 0 if the current page is not in the Content Linking List file.

ASP Content Linking Component GetNextDescription Method

The **GetNextDescription** method retrieves the text description of the next item in the Content Linking List file.

Arguments: ASP Content Linking Component GetNextDescription Method

listURL	The location of the Content Linking List file.

Return Values: ASP Content Linking Component GetNextDescription Method

The **GetNextDescription** method returns an ASCII string describing the next item in the Content Linking List file. If the current page is not found in the list file, **GetNextDescription** returns the string description of the last page on the list.

ASP Content Linking Component GetNextURL Method

The **GetNextURL** method retrieves the URL of the next item in the Content Linking List file.

Arguments: ASP Content Linking Component GetNextURL Method

listURL	The location of the Content Linking List file.

Return Values: ASP Content Linking Component GetNextURL Method

This method returns the URL of the next page specified in the Content Linking List file. If the current page is not specified in the Content Linking List file, **GetNextURL** returns the URL of the last page on the list.

Examples: ASP Content Linking Component GetNextURL Method

The following example uses the **GetNextURL** method to embed a link to the next page in the Content Linking List file. The advantage of using **GetNextURL** is that when you change the order or number of the content pages, you only have to update the list in the Content Linking List file and do not need to update the navigational links on each page.

```
<% Set NextLink = Server.CreateObject ("MSWC.NextLink") %>

<A HREF="<%= NextLink.GetNextURL ("/data/nextlink.txt") %>">Next
Page </A>
```

ASP Content Linking Component GetNthDescription Method

The **GetNthDescription** method retrieves a text description of the *nth* item in the Content Linking List file.

Arguments: ASP Content Linking Component GetNthDescription Method

listURL	The location of the Content Linking List file.
index	The index number of an item in the Content Linking List file.

Return Values: ASP Content Linking Component GetNthDescription Method

This method returns a string.

ASP Content Linking Component GetNthURL Method

The **GetNthURL** method returns the URL of the *N*th item in the Content Linking List file.

Arguments: ASP Content Linking Component GetNthURL Method

listURL	The location of the Content Linking List file.
index	The index number of an item in the Content Linking List file.

Return Values: ASP Content Linking Component GetNthURL Method

This method returns a string.

ASP Content Linking Component GetPreviousDescription Method

The **GetPreviousDescription** method retrieves a text description of the previous item in the Content Linking List file.

Arguments: ASP Content Linking Component GetPreviousDescription Method

listURL	The location of the Content Linking List file.

Return Values: ASP Content Linking Component GetPreviousDescription Method

This method returns a string describing either the previous item in the Content Linking List file or, if the current page is not in the file, the first item on the list.

ASP Content Linking Component GetPreviousURL Method

The **GetPreviousURL** method returns the URL of the previous item in the Content Linking List file.

Arguments: ASP Content Linking Component GetPreviousURL Method

listURL	The location of the Content Linking List file.

Return Values: ASP Content Linking Component GetPreviousURL Method

This method returns a string containing the URL of the previous item in the Content Linking List file. If the current page is not specified in the Content Linking List file, **GetPreviousURL** returns the URL of the first page in the file.

ASP Content Rotator Component

The Content Rotator component creates a **ContentRotator** object that automatically rotates HTML content strings on a Web page. Each time a user requests the Web page, the object displays a new HTML content string based upon information that you specify in a Content Schedule file.

Because the content strings can contain HTML tags, you can display any type of content that HTML can represent: text, images, or hyperlinks. For example, you can use this component to rotate through a list of daily quotations or hyperlinks, or to change text and background colors each time the Web page is opened.

Because the **ContentRotator** object uses a random generator to select which of the weighted content strings is displayed, a string may be repeated. This is most likely to occur if there are few entries in the Content Schedule file, or if one entry is weighted much higher than the others.

Registry Settings: ASP Content Rotator Component

The ASP Content Rotator component makes use of no registry settings.

Syntax: ASP Content Rotator Component

The Content Rotator component is registered with the ProgId of "MSWC.ContentRotator." The following VBScript excerpt shows creating an instance of the control.

```
Set NextTip = Server.CreateObject( "MSWC.ContentRotator" )
```

Properties: ASP Content Rotator Component

None

Methods: ASP Content Rotator Component

ChooseContent

GetAllContent

ASP Content Rotator Component Content Schedule File

The Content Schedule file contains information that the **ContentRotator** object uses to manage and display the specified content. In this file you include any number of HTML content string entries. Each entry consists of two parts: a line that begins with double percentage signs (%%) and contains both the relative weight and any comments, and a second part that contains the HTML content string itself.

Syntax: ASP Content Rotator Component Content Schedule File

```
%% [#Weight] [//Comments]
ContentString
```

Parameters: ASP Content Rotator Component Content Schedule File

ASP Content Rotator Component Content Schedule File Weight Parameter[0]

This optional parameter specifies a number between 0 and 10000 that indicates the relative weight of the HTML content string. The probability of a particular content string being displayed by the **ContentRotator** object can be expressed as the *Weight* of that content string divided by the sum of *Weight* values for all entries in the Content Schedule file.

For example, if a Content Schedule file contained three content strings with respective weights of 1, 3, and 4, the Content Rotator displays the first content string one-eighth of the time, the second string three-eighths of the time, and the third string half of the time.

A *Weight* of 0 will cause a content entry to be ignored.

If *Weight* is not specified, the default value is 1.

If the sum of all weight values exceeds 10000, an error will be generated when the schedule file is accessed by a call to either the **GetAllContent** or **ChooseContent** methods.

ASP Content Rotator Component Content Schedule File Comments Parameter

This optional parameter contains comments about the entry. These comments are for development use only and are not displayed to the user. If you require more than one line of comments, you must start each additional comment line with a line delimiter (%%) followed by a comment delimiter (//).

ASP Content Rotator Component Content Schedule File ContentString Parameter

The HTML content that the **ContentRotator** object displays. For example, you can present a line of text, an image, or a sound.

ContentString may include one or more lines. The **ContentRotator** object treats everything between blocks of double percent signs (%%) as a single HTML content string.

Examples: ASP Content Rotator Component Content Schedule File Parameters[0]

The following is an example of a Content Schedule file.

Note

Because the content strings can contain HTML tags, you can display any type of content that can be represented with HTML, including text, images, and hyperlinks.

```
-------------Content.txt--------------------
%% // Because no value is set for Weight, the default value is 1.
Don't run with scissors.
%% #2 // Content can be more than one line long.
%% // Additional line of comments.
%% // Yet another line of comments.
<FONT FACE="ARIAL,HELVETICA" SIZE="2">
 Let a
 <H1>smile</H1>
 be your umbrella.
</FONT>
%% #3 // This is our favorite image, so show it most often.
<IMG SRC="/images/happy.gif">
%%
Here's the <A HREF="secret.asp">secret link.</A>
```

ASP Content Rotator Component Methods

ASP Content Rotator Component ChooseContent Method[0]

The **ChooseContent** method retrieves an HTML content string from the Content Schedule file. The method retrieves a new content string each time the script is run, such as when a user opens or reloads a page.

Arguments: ASP Content Rotator Component ChooseContent Method

content-schedule-path

Specifies the location of the Content Schedule file.

This parameter can be specified either as a relative or virtual path. For example, if the Content Schedule file, Content.txt, and the .asp file that called **ChooseContent** both resided in the directory /MyApp/Tips/, where MyApp is a virtual directory on the server, then either the full virtual path (/MyApp/Tips/Content.txt) or the relative path (Content.txt) could be specified for *content-schedule-path*.

The **ContentRotator** object calls the **Server.MapPath** method to map the specified path to a physical directory. For more information, see the Server Object reference pages.

Return Value: ASP Content Rotator Component ChooseContent Method

Returns an HTML content string from the Content Schedule file.

Examples: ASP Content Rotator Component ChooseContent Method

The following example gets a new tip from the Content.txt file in the /Tips/ virtual directory.

```
<%
 Set NextTip = Server.CreateObject("MSWC.ContentRotator")
 Tip = NextTip.ChooseContent("/Tips/Content.txt")
 Response.Write Tip
%>
```

ASP Content Rotator Component GetAllContent Method

The **GetAllContent** method retrieves all of the HTML content strings from the Content Schedule file and writes them directly to the Web page as a list with an <HR> tag after each entry.

This method is typically used during authoring, to proofread the Content Schedule file.

Arguments: ASP Content Rotator Component GetAllContent Method

content-schedule-path

Specifies the location of the Content Schedule file.

This parameter can be specified either as a relative or virtual path. For example, if the Content Schedule file, Content.txt, and the .asp file that called **GetAllContent** both resided in the directory /MyApp/Tips/, where MyApp is a virtual directory on the server, then either the full virtual path (/MyApp/Tips/Content.txt) or the relative path (Content.txt) could be specified for content-schedule-path.

The **ContentRotator** object calls the **Server.MapPath** method to map the specified path to a physical directory. For more information, see the Server Object reference pages.

Remarks: ASP Content Rotator Component GetAllContent Method

The Content Rotator component uses the **Response.Write** method to write output directly to the .asp page that called the **GetAllContent** method. For more information, see the Response object topics.

Examples: ASP Content Rotator Component GetAllContent Method

The following example uses the **GetAllContent** method to display all of the entries in the Content Schedule file.

```
<H1>Tips Stored in the Content Schedule File:</H1>
<%
 Set Tips = Server.CreateObject("MSWC.ContentRotator")
 Tips.GetAllContent("/Tips/Content.txt")
%>
```

The preceding example produces HTML output such as the following:

```
<H1>Tips Stored in the Content Schedule File:</H1>
<HR>
Don’t run with scissors.
<HR>
<FONT FACE="ARIAL,HELVETICA" SIZE="2">
 Let a
 <H1>smile</H1>
 be your umbrella.
</FONT>
<HR>
<IMG SRC="/images/happy.gif">
<HR>
Here’s the <A HREF="secret.asp">secret link.</A>
<HR>
```

ASP Counters Component

The Counter component creates a **Counters** object that can create, store, increment, and retrieve any number of individual counters.

A counter is a persistent value that contains an integer. You can manipulate a counter with the **Get**, **Increment**, **Set**, and **Remove** methods of the **Counters** object. Once you create the counter, it persists until you remove it.

Counters do not automatically increment on an event like a page hit. You must manually set or increment counters using the **Set** and **Increment** methods.

Counters are not limited in scope. Once you create a counter, any page on your site can retrieve or manipulate its value. For example, if you increment and display a counter named *hits* in a page called Page1.asp, and you increment *hits* in another page called Page2.asp, both pages will increment the same counter. If you hit Page1.asp, and increment *hits* to 34, hitting Page2.asp will increment *hits* to 35. The next time you hit Page1.asp, *hits* will increment to 36.

All counters are stored in a single text file, counters.txt.

Only create one **Counters** object in your site. This single **Counters** object can create any number of individual counters.

Registry Settings: ASP Counters Component

The Counters Component makes use of no registry settings.

Syntax: ASP Counters Component

The Counters control is registered with the ProgId of "MSWC.Counters." Create the **Counters** object one time on your site by adding the following to the Global.asa file:

```
<OBJECT
RUNAT=Server
SCOPE=Application
ID=Counter
PROGID="MSWC.Counters">
</OBJECT>
```

Properties: ASP Counters Component

None.

Methods: ASP Counters Component

- Get
- Increment
- Remove
- Set

ASP Counters Component Methods

ASP Counters Component Get Method

The **Get** method takes the name of a counter and returns the current value of the counter. If the counter doesn't exist, the method creates it and sets it to 0.

Arguments: ASP Counters Component Get Method

CounterName	A string containing the name of the counter.

Examples: ASP Counters Component Get Method

Display the value a counter with `<%= Counters.Get(CounterName) %>`. Assign the value of the counter to a variable with `<% countervar = Counters.Get(CounterName) %>`.

The following script displays the vote tally from a poll about favorite colors.

```
<%
 If colornumber = "1" Then
  Counters.Increment("greencounter")
 Else
  If colornumber = "2" Then
   Counters.Increment("bluecounter")
  Else
   If colornumber = "0" Then
    Counters.Increment("redcounter")
   End If
  End If
 End If
%>
<P>Current vote tally:
<P>red: <% =Counters.Get("redcounter") %>
<P>green: <% = Counters.Get("greencounter") %>
<P>blue: <% = Counters.Get("bluecounter") %>
```

ASP Counters Component Increment Method

The **Increment** method takes the name of a counter, adds 1 to the current value of the counter, and returns the counter's new value. If the counter doesn't exist, the method creates it and sets its value to 1.

Arguments: ASP Counters Component Increment Method

CounterName	A string containing the name of the counter.

Examples: ASP Counters Component Increment Method

Increment the value of a counter with `<% Counters.Increment(CounterName) %>`. Increment and display the value of a counter with `<%= Counters.Increment(CounterName) %>`.

To retrieve the value of a counter, use **Counters.Get**. To set a counter to a specific value, use **Counters.Set**.

The following code implements a one-line page-hit counter.

```
<P>There have been <%= Counters.Increment("hits") %> visits to this
Web page. </P>
```

In this example, **Counters.Increment** increases the counter by one each time the client requests the page from the server.

ASP Counters Component Remove Method

The **Remove** method takes the name of a counter, removes the counter from the **Counters** object, and deletes the counter from the Counters.txt file.

Arguments: ASP Counters Component Remove Method

CounterName	A string containing the name of the counter.

Examples: ASP Counters Component Remove Method

The following code removes the counter `hitscounter` from the counters.txt file.

```
<% Counters.Remove(hitscounter) %>
```

ASP Counters Component Set Method

The **Set** method takes the name of a counter and an integer, sets the counter to the value of the integer, and returns the new value. If the counter doesn't exist, **Counters.Set** creates it and sets it to the value of the integer.

Arguments: ASP Counters Component Set Method

CounterName	A string containing the name of the counter.
int	The new integer value for *CounterName*.

Examples: ASP Counters Component Set Method

The following code resets the hit counter `pageHits` to 0:

```
<% Counters.Set(pageHits, 0) %>
```

ASP MyInfo Component

The MyInfo component creates a **MyInfo** object that keeps track of personal information, such as the site administrator's name, address, and display choices. Typically, the administrator types this information directly into the Web server interface. However, you can set the values of the properties directly by using a script in an ASP page.

Note

Sun Chili!Soft ASP does not implement the default properties available under Windows Personal Web Services.

Each property of a **MyInfo** object returns a string. If a **MyInfo** property has no value set, the property returns an empty string.

Create new **MyInfo** properties for values that remain consistent throughout a site. You can create new **MyInfo** properties by simply assigning a string value to them. The following example creates the new properties **DogName** and **DogBreed**. These new properties are stored persistently along with the other **MyInfo** properties.

```
<%
 MyInfo.DogName = "Snoopy"
 MyInfo.DogBreed = "Beagle"
%>
```

The values of **MyInfo** properties are stored in the text file, libmyinfo.ini. On UNIX systems, this file is located in [C-ASP_INSTALL_DIR]/server/lib/sunos5_optimized, where [C-ASP_INSTALL_DIR] is the complete directory path of the Sun Chili!Soft ASP installation directory. On Linux systems, this file is located in [C-ASP_INSTALL_DIR]/lib-chilicom.

The Sun Chili!Soft ASP implementation of the MyInfo component is compatible with the MyInfo.xml file produced by the Microsoft implementation; however, Microsoft implements the text file as an XML file, while Sun Chili!Soft ASP does not.

Registry Settings: ASP MyInfo Component

The control makes use of no registry settings.

Syntax: ASP MyInfo Component

The MyInfo component is registered with the ProgID of "MSWC.MyInfo."

The following code in the global.asa file creates one instance of the **MyInfo** object. In this example, the object is given Session scope, but a **MyInfo** object could also be given Application scope:

```
<OBJECT
RUNAT=Server
SCOPE=Session
ID=MyInfo
```

```
PROGID="MSWC.MyInfo">
</OBJECT>
```

Properties: ASP MyInfo Component

The Chili!Soft implementation does not implement the default properties available under Windows Personal Web Services. You create your own properties as described in this topic.

Methods: ASP MyInfo Component

None.

ASP Tools Component

The Tools component creates a **Tools** object that provides utilities that enable you to easily add sophisticated functionality to your Web pages.

Registry Settings: ASP Tools Component

The control makes use of no registry settings.

Syntax: ASP Tools Component

The Tools component is registered with the ProgId of "MSWC.Tools." The following VBScript excerpt shows creating an instance of the control.

```
Set Tools = Server.CreateObject( "MSWC.Tools" )
```

The Tools component exposes the following properties and methods.

Properties: ASP Tools Component

None.

Methods: ASP Tools Component

- FileExists
- Owner
- ProcessForm
- PluginExists
- Random

ASP Tools Component Methods

ASP Tools Component FileExists Method

The **FileExists** method checks the existence of a file. It returns –TRUE if the specified URL exists within a published directory. If the file does not exist, it returns FALSE.

Arguments: ASP Tools Component FileExists Method

URL	A string that specifies the relative URL of the file you are checking.

Remarks: ASP Tools Component FileExists Method

FileExists only checks the existence of files published on your site. Therefore, it takes a relative URL rather than an absolute URL.

Examples: ASP Tools Component FileExists Method

The following example demonstrates using the **FileExists** property to create a link if a particular file is present.

```
<%If Tools.FileExists("ie_animated.gif") then %>

 <p> <a HREF="/isapi/gomscom.asp?TARGET=/ie/"><img
src="ie_animated.gif"></a>

<% End If %>
```

ASP Tools Component Owner Method

The Sun Chili!Soft ASP implementation of this method always returns 0.

ASP Tools Component PluginExists Method

The Sun Chili!Soft ASP implementation of this method always returns 0.

ASP Tools Component ProcessForm Method

The **ProcessForm** method processes the contents of a form that has been submitted by a visitor to the Web site.

Arguments: ASP Tools Component ProcessForm Method

OutputFileURL	A string containing the relative URL of the file to which the processed data is written.
TemplateURL	A string containing the relative URL of the file that contains the template, or instructions, for processing the data.
InsertionPoint	An optional parameter indicating where in the output file to insert the process data. This parameter has not been implemented. If you include a value for this parameter it will be ignored.

Remarks: ASP Tools Component ProcessForm Method

The template files can contain ASP scripts. A script between <% and %> delimiters is treated just like other text in the template and copied into the output file. If the output file is an ASP document, the script will run when the output file is accessed. Scripts in template files can also be put between special <%% and %%> delimiters which cause the script to execute while **Tools.ProcessForm** is executing. Since these scripts are executed before the template data is saved in the output file, the results get saved in the output file, usually as standard text.

The scripts can use any of the ASP intrinsics for the page containing the script executing the **ProcessForm** method except for the **Response** objects. Instead, the miniscripts have their own **Response** objects with implementations of Write and BinaryWrite that write to the output file instead of the Web server output stream.

If the specified output file does not exist, the server creates it.

If the *InsertionPoint* parameter does not exist, **Tools.ProcessForm** replaces the entire output file. If the *InsertionPoint* parameter exists, and does not begin with an asterisk (*), **Tools.ProcessForm** finds the *InsertionPoint* string in the output file and inserts the data immediately after it. If the *InsertionPoint* string begins with an asterisk (*), **Tools.ProcessForm** finds the *InsertionPoint* string in the output file and inserts the data immediately before it. If the *InsertionPoint* string exists, but is not found in the output file, the data is appended to the end of the file.

Examples: ASP Tools Component ProcessForm Method

The following code demonstrates calling an .asp file to process a form.

```
<%

Tools.processform("/$Received
Messages/default.asp","MessageInsert.process","<SPAN>*")

%>
```

ASP Tools Component Random Method

The **Random** method returns an integer between –32768 to 32767.

Arguments: ASP Tools Component Random Method

None.

Remarks: ASP Tools Component Random Method

This method is similar to the **Rnd** function, but returns an integer.

To get a positive random integer, use the **Abs** function.

To get a random integer below a specific value, use the **Mod** function.

Examples: ASP Tools Component Random Method

```
<% = Tools.Random %> will display a random integer between -32768 to
32767. For example, -13067.
```

```
<% = ( Abs( Tools.Random ) ) %> will display a positive random
integer. For example, 23054.

<% = ( Abs( Tools.Random ) ) Mod 100 %> will display a random
integer between 0 and 99. For example, 63.
```

Chili!Beans Component Reference

The Chili!Beans ActiveX control is a wrapper that enables Java objects to be used by Component Object Model (COM) controllers (such as ActiveX scripting engines like VBScript). The control is designed to work with Java Virtual Machine versions 1.2 or greater.

To use Chili!Beans, a Java runtime environment (JRE) must be installed on the machine, and Chili!Beans must be enabled from the Administration Console. JRE 1.3.1 is included with Sun Chili!Soft ASP and is the recommended version. For more information, see "Enabling Java Support" in "Chapter 2: Installing and Configuring Sun Chili!Soft ASP," and "Enabling Chili!Beans" in this section.

Note

When using Chili!Beans with Sun Chili!Soft ASP for Solaris, you must use the JVM Native Threads.

In rare cases it is necessary to supply startup settings to the Java Virtual Machine. See "Supplying Java Virtual Machine Settings" in this section.

Chili!Beans is not available for the Cobalt or Windows versions of Sun Chili!Soft ASP.

This section provides information about enabling and disabling Chili!Beans, and reference information about using the component:

- Enabling Chili!Beans
- Using Null Objects with Chili!Beans
- Iterating a Collection with Chili!Beans
- Accessing Methods and Fields with Chili!Beans
- Limitations of Chili!Beans Objects
- Supplying Java VM Settings
- Constructing Java Objects with Chili!Beans

Enabling Chili!Beans

On UNIX and Linux platforms, Chili!Beans is enabled or disabled from the **Components** page in the Sun Chili!Soft ASP Administration Console. To use Chili!Beans, a Java runtime environment

(JRE) must be installed on the machine (see "Enabling Java Support" in "Chapter 2: Installing and Configuring Sun Chili!Soft ASP"), and Chili!Beans must be enabled.

Note

Chili!Beans is not available on Windows systems.

When Chili!Beans is enabled, you have the option of enabling or disabling the Java Virtual Machine (VM) Security Manager (the Java VM Security Manager is enabled by default). If the Java VM Security Manager is enabled, its default behavior is to prevent any access to system resources other than read-only access to the current directory. If the Java VM Security Manager is disabled, Java code executed by the Chili!Bean will run with unrestricted access to the File System and other system resources.

Note

For security reasons, the Java VM Security Manager should be enabled in multi-user environments in which users supply their own Java classes.

To selectively grant other privileges to Java code running in the Chili!Bean, with Java VM Security Manager enabled, use policytool to change the Virtual Machine's security settings as specified in the Java2 Security documentation.

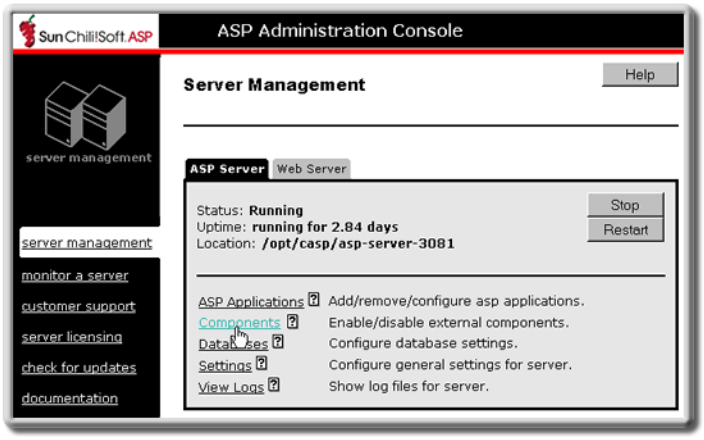
To enable or disable Chili!Beans

1. If necessary, open the Administration Console by using the following URL:

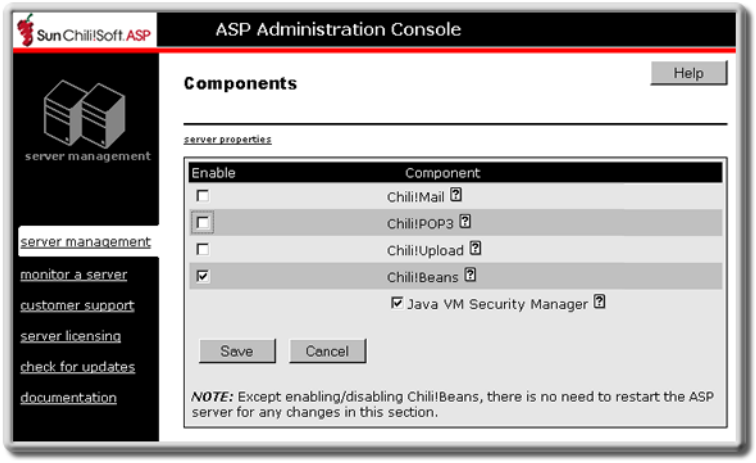
   http://[HOSTNAME]:[PORT]

   where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

2. On the **ASP Server** tab of the **Server Management** page (the first page to display when you open the Administration Console), click **Components**.



   The **Components** page displays.



3. On the **Components** page, click to select or clear the **Chili!Beans** check box.

   If the **Chili!Beans** box is selected, the **Java VM Security Manager** check box displays (this box is selected by default). Select or clear this box to enable or disable the Java VM Security Manager.

   Note: If you did not enable Chili!Beans support during installation (install a Java runtime environment), **Chili!Beans** will not be listed on the **Components** page. See "Enabling Java Support" in "Chapter 2: Installing and Configuring Sun Chili!Soft ASP."

4. Click **Save** to save your changes.

   – or –

   Click **Cancel** to revert to the last settings that were saved.

   The **Server Management** page displays.

5. To put your changes into effect, restart the ASP Server by clicking **Restart**.

Note

Restarting the ASP Server resets all **Session** and **Application** variables.

Using Null Objects with Chili!Beans

When a Chili!Beans-wrapped Java method returns a **Null** object, the **Null** object is translated to the special value *Nothing* when returned to the ASP script. If the special value *Nothing* is passed from the ASP script to a Chili!Bean, it is converted to a **Null** object before being passed to the Java method.

Iterating a Collection with Chili!Beans

If the Java object underlying a Chili!Beans control implements the java .util.Enumeration interface it will function like a COM Collection class, and you can use the **For…Each…Next** statement to iterate the Java object.

Accessing Methods and Fields with Chili!Beans

All public methods of a class are accessible from their **Chili!Beans** wrapper. If a class has multiple methods with the same name, the control will resolve the correct method at run time based on the arguments passed. In some cases the mappings of **Variant** data types in client scripts to Java data types can result in incorrect resolution between methods with similar signatures. The Chili!Beans control does not distinguish between methods or fields whose names differ only by case.

Uncaught exceptions thrown by Java method calls are caught by the control and reported to the controller as COM exceptions whose **Description** field is the **toString()** value of the Java **Exception** object thrown. If the CB_STACKTRACE environment variable is set to 1, a full stack trace for the exception is included in the description field. With Sun Chili!Soft ASP as the controller, this string is reported as part of the run-time error text and will appear in the browser.

Limitations of Chili!Beans Objects

The following limitations apply to **Chili!Beans** objects:

- A **Chili!Beans** object is accessible to all threads in a Sun Chili!Soft ASP application, and is thread-safe if the underlying Java class is thread-safe. The **Chili!Beans** object is marked in the registry with `ThreadingModel=both`; this means that **Chili!Beans** objects stored as **Application** or **Session** variables will be accessed from multiple threads and will certainly fail if their underlying Java code is not thread-safe.
- There is no support for multiple-dimension arrays as either arguments or return values.

Supplying Java Virtual Machine Settings

In very rare cases you must supply startup settings to the Java Virtual Machine. In a stand-alone Java application these settings are passed as command-line arguments.

Note

The mechanism described here is for expert Sun Chili!Soft ASP users only.

The startup settings can be passed to the Java Virtual Machine run by the Chili!Bean by specifying them in a configuration file. The default path to the file is as follows:

<CASP_INSTALL_DIRECTORY>/bean/bean.properties

This path can be customized by exporting the CB_PROPERTIES environment variable in the javasetup.sh script to the desired path.

Each line in the configuration file will be passed as an argument to the Java Virtual Machine at startup. For example, configuring the Chili!Bean with file:

```
#bean.properties

-Dfoo=bar

-Xint
```

has the same effect as starting the Java Virtual Machine from the command line with the command:

```
Java –Dfoo=bar –Xint <classname>
```

The meanings of individual arguments vary with Virtual Machine versions; consult the Virtual Machine's documentation.

Note

Never use this mechanism to change the startup classpath, unless all of the directories and JAR files set in the CLASSPATH by the javasetup.sh script are included.

Constructing Java Objects with Chili!Beans

The Chili!Beans control is used in client scripts in the same way that Microsoft implements COM wrappers for Java objects with the Microsoft JVM.

An instance of any Java class located on the path in the local CLASSPATH environment variable can be constructed, any public methods of the resulting object can be called, and any of its public fields can be accessed.

There are three ways to create a Java object by using Chili!Beans, as discussed in the following topics:

- Accessing a Java Class via Chili!Beans
- Registering a Java Class as a COM Component on Linux and UNIX
- Returning a Java Class From a Method Call or Field Access

Accessing a Java Class via Chili!Beans

You can use the **Chili!Beans** object **Construct** method to create instances of Chili!Beans. The first argument is the fully qualified name of the class to be instantiated, and the remaining arguments are the arguments to be passed to the desired constructor for that class. For example, if the package Database contains Table .class, the following script will create a **Table** object:

```
Set factory = Server.CreateObject("Chili.Beans")

set table = factory.Construct "Database/Table", "Employees",
CLng(100)
```

This will create an object named "table," using the constructor whose signature is

constructor(String, Int)

If the class name cannot be found on the CLASSPATH or if there is no public constructor whose signature matches the arguments passed to **Construct**, a run-time error occurs in the client script.

To use a Java class with Sun Chili!Soft ASP, the .class file must exist in a directory that is listed in the Java CLASSPATH environment variable, or it must be registered with Sun Chili!Soft ASP as described in "Registering a Java Class as a COM Component on Linux and UNIX" in this chapter.

Registering a Java Class as a COM Component on Linux and UNIX

The chregclass tool included with Sun Chili!Soft ASP enables you to register a Java class as a COM component on Linux and UNIX. You register a Java class by using the chregclass tool to create a registry entry that maps a given ProgID to the Java class. The chregclass tool is similar to the javareg tool provided for the Microsoft JVM.

Notes

In order to register a Java class to use with Sun Chili!Soft ASP, the .class file must exist in a directory that is listed in the Java CLASSPATH environment variable.

Any class registered by using chregclass must have a public default constructor to instantiate the class. This applies to all chregclass calls.

To register a Java class as a COM component

1. Log in as root and change directories to the Sun Chili!Soft ASP installation directory.
2. Stop the ASP Server, as described in "Stopping and Restarting the ASP Server" in "Chapter 3: Managing Sun Chili!Soft ASP."
3. Map the ProgID to the Java class by running the following command:

   ```
   chregclass [-f] [ProgID] [JAVA_CLASS]
   ```

   where `[ProgID]` is the Prog ID you want to map and `[JAVA_CLASS]` is the name of the Java class you want to register. `[JAVA_CLASS]` should not include the .class extension. If it does, the mapping will not work.
4. Restart the ASP Server, as described in "Stopping and Restarting the ASP Server" in "Chapter 3: Managing Sun Chili!Soft ASP."

For example, to register the Table .class in the package Database on the CLASSPATH, use the following command:

```
chregclass Db.Table Database/Table
```

After running this command, you can then construct a **Table** object in a client script as follows:

```
set table = Server.CreateObject("Db.Table")
```

Returning a Java Class from a Method Call or Field Access

A Java object returned from a Java method call or field access in a client script is automatically wrapped in its own Chili!Beans wrapper. For example, the classes Table and Record are defined as:

```
//Table.java
package Database ;
public class Table {
  public Table(String name, in initialSize) {...};
  public int numRecords() {...};
  public Record getEmployee(int employeeNumber)
{...}
  ...
}
//Record.java
package Database ;
public class Record {
  public Record() {...};
  public String m_LastName ;
  public String m_FirstName ;
  ...
}
```

The following ASP script will print the names of the employees in Table:

```
set t = Server.CreateObject("Chili.Beans")
t.ClassName = "Database/Table"
t.Construct "Employees", 100
for I = 0 to t.numRecords - 1
  set record = t .getEmployee(I)
  Response.Write(record.m_FirstName & " " _
   & record .m_LastName & "<br>")
next
```

The Java objects returned by the getEmployee calls on the **Table** object are automatically given Chili!Beans wrappers and their methods and fields are available even though they have not been constructed with the **CreateObject** method.

Component Programmer's Reference

On Windows and UNIX systems, Sun Chili!Soft ASP provides support for custom server components, which can be useful when your Web applications require complex business logic. The application developer may find it more efficient to encapsulate this business logic in a custom server component written in an advanced language like C++ (on Windows) or Java (on UNIX), rather than trying to implement it with script. Sun Chili!Soft ASP uses COM (Component Object Model) as the standard interface for creating custom components.

Note

At this time there is no compiler or API for third-party COM objects available for UNIX.

On Windows, Sun Chili!Soft ASP server components can be written in any language that supports COM, including Visual Basic, C++, and Java. On UNIX, Sun Chili!Soft ASP server components can be written in Java. If you wish to develop your components in Java, Chili!Beans shields you from many of the details of COM. For more information, see "Chili!Beans Component Reference" in this chapter.

If you want to develop components using C++, this section is for you.

This section provides the following reference information about components:

- Scope and Threading
- C++ Interfaces Reference
- IApplicationObject Interface
- IReadCookie Interface
- IRequest Interface
- IRequestDictionary Interface
- IResponse Interface
- IScriptingContext Interface
- IServer Interface
- ISessionObject Interface
- IStringList Interface
- IVariantDictionary Interface
- IWriteCookie Interface
- COM on UNIX

Scope and Threading

COM objects used in ASP pages can have Page, Session, or Application scope depending on how and when they are created. Objects created on ASP pages will have Page scope by default; they are created when the page is processed, and are released when page processing is complete. Objects with Page scope are only available on the page where they are created.

Objects given Session scope or Application scope in the global.asa file or on an ASP page are then available to other ASP pages. Objects that are given Session or Application scope using the **Server.CreateObject** method must be marked as both apartment and free-threaded. It is possible to create single, apartment, and free-threaded objects in the global.asa file using the extended `<OBJECT>` tag syntax, but doing so will constrain the Sun Chili!Soft ASP server to a single thread, significantly impacting performance.

For maximum flexibility, all COM objects used on ASP pages should be marked as "Both," and use both the apartment and free-threaded models.

C++ Interfaces Reference

Sun Chili!Soft ASP implements interfaces that allow your components to access the properties and methods of built-in ASP objects. You component can use these object interfaces to access the properties, methods, and collections of the built-in objects.

Sun Chili!Soft ASP does not currently support the **IObjectContext** interface required to access Microsoft Transaction Server methods. The **IScriptingContext** should be used to access the built-in object interfaces.

The following table lists the built-in object interfaces:

Interface	Description
IApplicationObject	Calls the methods and properties of the **Application** object.
IReadCookie	Retrieves the values stored in the read-only **Cookies** collection.
IRequest	Calls the methods and properties of the **Request** object.
IRequestDictionary	Indexes the collections of the **IRequest** interface.
IResponse	Calls the methods and properties of the **Response** object.
IScriptingContext	Returns a pointer to the interface on one of the built-in objects: **IApplicationObject**, **IRequest**, **IResponse**, **IServer**, or **ISessionObject**.
IServer	Calls the methods and properties of the **Server** object.
ISessionObject	Calls the methods and properties of the **Session** object.
IStringList	Retrieves the values stored in a string list, used in the **QueryString**, **Form** or **ServerVariables** collections.
IVariantDictionary	Retrieves the items stored in the **Application** and **Session**

	Contents and **StaticObjects** collections.
IWriteCookie	Sets the values and attributes in the write-only **Cookies** collection.

To use **IScriptingContext** and object interfaces in a C++ component, you must include the header file asptlb.h.

For more information, see "ASP Built-in Objects Reference" in this chapter.

COM on UNIX

Sun Chili!Soft ASP uses an implementation of COM for UNIX developed internally. At this time there is no compiler or API available to port custom objects to UNIX platforms. If you are developing your components in Java, you can use Sun Chili!Soft Chili!Beans technology instead.

IApplicationObject Interface

The **IApplicationObject** interface exposes the methods of the **Application** object.

IApplicationObject::get_Contents	Retrieves the **Contents** collection
IApplicationObject::get_StaticObjects	Retrieves the **StaticObjects** collection.
IApplicationObject::get_Value	Retrieves the value of a variable stored in the **Application** object.
IApplicationObject::Lock	Prevents other clients from accessing the variables stored in the **Application** object until **IApplication::Unlock** is called.
IApplicationObject::putref_Value	Stores a variable in the **Application** object by reference.
IApplicationObject::put_Value	Stores a variable in the **Application** object by value.
IApplicationObject::UnLock	Releases the lock that was created by the **IApplication::Lock** method.

IApplication also supports all **IUnknown** and **IDispatch** interface methods.

IApplicationObject::get_Contents

The **IApplicationObject::get_Contents** method retrieves the **Application.Contents** collection.

```
HRESULT get_Contents(

  IVariantDictionary **ppProperties
```

)

Parameters: IApplicationObject::get_Contents

ppProperties

Points to an **IVariantDictionary** interface pointer that receives the **Contents** collection.

Remarks: IApplicationObject::get_Contents

The **Application.Contents** collection contains all variables and objects that have been given application scope with the **Server.CreateObject** command. You can iterate through the **Contents** collection with the **IVariantDictionary::get_NewEnum** method. You can also retrieve a specific item with the **IVariantDictionary::get_Item** method.

IApplicationObject::get_StaticObjects

The **IApplicationObject::get_StaticObjects** method retrieves the **Appliction.StaticObjects** collection.

```
HRESULT get_StaticObjects(

  IVariantDictionary **ppProperties
```

);

Parameters: IApplicationObject::get_StaticObjects

ppProperties

Points to an **IVariantDictionary** interface pointer that receives the **StaticObjects** collection.

Remarks: IApplicationObject::get_StaticObjects

You can iterate through the **StaticObjects** collection with the **IVariantDictionary::get_NewEnum** method. You can also retrieve a specific item with the **IVariantDictionary::get_Item** method.

IApplicationObject::get_Value

The **IApplicationObject::get_Value** method retrieves the value of a variable stored in the **Application** object.

```
HRESULT get_Value

  BSTR bstrValue,

  VARIANT * pvar
```

)

Parameters: IApplicationObject::get_Value

bstrValue

A binary string that contains the name of the variable to retrieve.

pvar

Points to a variant that holds the value for the variable specified in *bstrValue*.

Remarks: IApplicationObject::get_Value

You can use this method to access variables that have been given application scope.

IApplicationObject::Lock

The **IApplicationObject::Lock** method prevents other clients from accessing the variables stored in the **Application** object until the **IApplicationObject::UnLock** method has been called.

HRESULT Lock(VOID);

IApplicationObject::putref_Value

The **IApplicationObject::putref_Value** method stores an object in the **Application** object. Equivalent to the **IApplicationObject::put_Value** method for non-objects.

```
HRESULT putref_Value(

  BSTR bstrValue,

  VARIANT var
);
```

Parameters: IApplicationObject::putref_Value

bstrValue
A binary string that contains the name of the object to store.

var

A variant that contains the reference to the object.

Remarks: IApplicationObject::putref_Value

You can use this method to create a reference to an object. Giving some object application scope will have a negative impact on server performance. See Scope and Threading for more information.

IApplicationObject::put_Value

The **IApplicationObject::put_Value** method stores a variant in the **Application** object.

```
HRESULT put_Value(
```

```
    BSTR bstrValue,

    VARIANT var
);
```

Parameters: IApplicationObject::put_Value

bstrValue

A binary name that contains the name of the variable.

var

A variant that contains the variable value.

Remarks: IApplicationObject::put_Value

If the variant being stored is an object, Sun Chili!Soft ASP will attempt to store the default value of that object into the application. If Sun Chili!Soft ASP cannot get the default value, the call will fail.

IApplicationObject::UnLock

The **IApplicationObject::UnLock** method releases a variable that was locked by the **IApplicationObject::Lock** method.

HRESULT UnLock(VOID)

IReadCookie Interface

You can use the **IReadCookie** interface to access the objet returned from the read-only **Cookies** collection. Calling **IRequestDictionary::get_Item** on the read-only **Cookies** collection will always return an object that implements the **IReadCookie** interface.

IReadCookie::get_HasKeys	Retrieves a Boolean indicating whether the cookie has keys.
IReadCookie::get_Item	Retrieves the specified item from a cookie dictionary.
IReadCookie::get_NewEnum	Retrieves an enumerator for the **Cookies** collection.

IReadCookie and **IWriteCookie** are interfaces for the same object. If you have an **IReadCookie** pointer, you can use the **IUnknown::QueryInterface** method on an **IWriteCookie** pointer.

IReadCookie::get_HasKeys

The **IReadCookie::get_HasKeys** method returns a Boolean value which indicates whether or not the cookie has keys.

```
HRESULT get_HasKeys(

  VARIANT_BOOL * pfHasKeys

);
```

Parameters: IReadCookie::get_HasKeys

pfHasKeys

Points to a Boolean that indicates whether or not the cookie has keys.

Remarks: IReadCookie::get_HasKeys

If the cookie has been implemented as a collection, this method will return TRUE. If the cookie has keys, use the **IReadCookie::get_NewEnum** method to retrieve an enumerator for iterating through the collection.

IReadCookie::get_Item

The **IReadCookie::get_Item** method retrieves the specified item from a **Cookie** object.

```
HRESULT get_Item(

  VARIANT Var,

  VARIANT * pVariantReturn

);
```

Parameters: IReadCookie::get_Item

Var

A variant that contains the name of the item in the collection.

pVariantReturn

Points to a variant that receives the item value.

Remarks: IReadCookie::get_Item

If *Var* is a string, the cookie is assumed to be a dictionary cookie and *Var* is treated as the key. If *Var* is a variant with type `VT_ERROR` and error code `DISP_E_PARAMNOTFOUND`, then the cookie is assumed to be a simple cookie and the primary value is returned. (If the cookie is a dictionary, the sequence of key/value pairs are URL-encoded and returned.)

If a dictionary lookup is performed and the item is not found, then *pVariantReturn* returns `VT_EMPTY`. Otherwise a `VT_BSTR` is always returned.

IReadCookie::get_NewEnum

The **IReadCookie::get_NewEnum** method returns an enumerator interface which can be used to iterate through the items in a cookie.

```
HRESULT get_NewEnum(

  IUnknown ** ppEnumReturn
);
```

Parameters: IReadCookie::get_NewEnum

ppEnumReturn

Points to an **IUnknown** interface pointer that receives the enumerator.

Remarks: IReadCookie::get_NewEnum

If a cookie has been implemented as a collection, you can use this method to iterate through all the members of its collection.

IRequest Interface

The **IRequest** interface exposes methods that access the collections of the **Request** object.

IRequest::BinaryRead	Retrieves the current request and places it into a safe array.
IRequest::get_Cookies	Retrieves the **Cookies** collection.
IRequest::get_Form	Retrieves the **Form** collection.
IRequest::get_Item	Retrieves an item from the **Request** collection by looking for the first match in the **QueryString**, **Form**, **Cookies**, **ClientCertificate**, and **ServerVariables** collections.
IRequest::get_QueryString	Retrieves the **QueryString** collection.
IRequest::get_ServerVariables	Retrieves the **ServerVariables** collection.
IRequest::get_TotalBytes	Returns the size of the current request in bytes.

The **IRequest** interface also supports the **IUnknown** and **IDispatch** interface methods.

The Sun Chili!Soft ASP **IRequest** interface does not currently support the **IClientCertificate** collection.

IRequest::BinaryRead

The **IRequest::BinaryRead** method retrieves the current **Request** object in a safe array.

```
HRESULT BinaryRead(

  VARIANT *pvarCountToRead,

  VARIANT *pvarReturn
```

);

Parameters: IRequest::BinaryRead

pvarCountToRead

Points to a variant that contains an integer value of the bytes to read.

pvarReturn

Points to a safe array that contains the bytes that were read by an HTTP `POST`.

Remarks: IRequest::BinaryRead

A safe array is an array that contains information about the number of dimensions and the upper and lower bounds of the dimensions.

IRequest::get_Cookies

The **IRequest::get_Cookies** method retrieves the **Cookies** collection of the **Request** object.

HRESULT get_Cookies(

IRequestDictionary ** *ppDictReturn*

);

Parameters: IRequest::get_Cookies

ppDictReturn

Points to an **IRequestDictionary** interface pointer that receives the **Cookies** collection.

Remarks: IRequest::get_Cookies

You can iterate through the **Cookies** collection with the **IRequestDictionary::get_NewEnum** method or retrieve a specific cookie with the **IRequestDictionary::get_Item** method.

IRequest::get_Form

The **IRequest::get_Form** method retrieves the **Form** collection of the **Request** object.

```
HRESULT get_Form(

  IRequestDictionary ** ppDictReturn
```

);

Parameters: IRequest::get_Form

ppDictReturn

Points to an **IRequestDictionary** interface that receives the **Form** collection.

Remarks: IRequest::get_Form

You can iterate through the **Form** collection with the **IRequestDictionary::get_NewEnum** method or retrieve a specific cookie with the **IRequestDictionary::get_Item** method.

IRequest::get_Item

The **IRequest::get_Item** method retrieves a pointer to an interface pointer of the first item in the **Request** object's collections that contains the specified string.

```
HRESULT get_Item(

  BSTR bstrVar,

  IDispatch ** ppObjReturn
);
```

Parameters: IRequest::get_Item

bstrVar

A binary string that contains the name of the item to retrieve.

ppObjReturn

Points to an **IDispatch** pointer that receives the object that contains the value specified in *bstrVar*.

Remarks: IRequest::get_Item

The **Request** object's collections are searched in the following order: **QueryString**, **Form**, **Cookies**, **ClientCertificate**, and **ServerVariables**.

If the object containing *bstrVar* is found in the **QueryString**, **Form**, or **ServerVariables** collection, then *ppObjReturn* points to an object that supports the **IStringList** interface.

If the object is found in the **Cookies** collection, *ppObjReturn* points to an object that supports the **IReadCookie** interface.

IRequest::get_QueryString

The **IRequest::get_QueryString** method retrieves the **QueryString** collection of the **Request** object.

```
HRESULT get_QueryString(

  IRequestDictionary ** ppDictReturn
);
```

Parameters: IRequest::get_QueryString

ppDictReturn

Points to an **IRequestDictionary** interface pointer that receives the **QueryString** collection.

Remarks: IRequest::get_QueryString

You can iterate through the **QueryString** collection with the **IRequestDictionary::get_NewEnum** method or retrieve a specific cookie with the **IRequestDictionary::get_Item** method.

IRequest::get_ServerVariables

The **IRequest::get_ServerVariables** method retrieves the **QueryString** collection of the **Request** object.

```
HRESULT get_ServerVariables(

  IRequestDictionary ** ppDictReturn
);
```

Parameters: IRequest::get_ServerVariables

ppDictReturn

Points to an **IRequestDictionary** interface pointer that receives the **ServerVariables** collection.

Remarks: IRequest::get_ServerVariables

You can iterate through the **ServerVariables** collection with the **IRequestDictionary::get_NewEnum** method or retrieve a specific cookie with the **IRequestDictionary::get_Item** method.

IRequest::get_TotalBytes

The **IRequest::get_TotalBytes** method retrieves the size of the current request in bytes.

```
get_TotalBytes(

  LONG pcbTotal
);
```

Parameters: IRequest::get_TotalBytes

pcbTotal

Points to a long integer that contains the size of the current request in bytes.

IRequestDictionary Interface

The **IRequestDictionary** interface is a general interface wrapper that can wrap the following collections:

- Request.Cookies

- Request.Form
- Request.QueryString
- Request.ServerVariables
- Response.Cookies

The behavior of the **IRequestDictionary** interface depends on how the Request or Response object implements the **IRequestDictionary::get_Item** method.

IRequestDictionary::get_Count	Retrieves the number of items in a dictionary.
IRequestDictionary::get_Item	Retrieves the specified item from a dictionary.
IRequestDictionary::get_Key	Retrieves the identifier of the item to be retrieved from the collection.
IRequestDictionary::get_NewEnum	Retrieves an enumerator for the collection.

The **IRequestDictionary** interface also supports the **IUnknown** and **IDispatch** interface methods.

IRequestDictionary::get_Count

The **IRequestDictionary::get_Count** method determines the total number of items in a collection.

```
HRESULT STDMETHODCALLTYPE get_Count(
  int * cStrRet
);
```

Return Values: IRequestDictionary::get_Count

cStrRet

Points to an integer that contains the total number of items in a collection.

IRequestDictionary::get_Item

The **IRequestDictionary::get_Item** method retrieves the specified item from a **Request** object dictionary

```
HRESULT get_Item(
  VARIANT Var,
  VARIANT * pVariantReturn
);
```

Parameters: IRequestDictionary::get_Item

Var

A variant that contains the name of the item in the collection.

pVariantReturn

Points to a variant that receives the item.

Remarks: IRequestDictionary::get_Item

Objects can specify that one or more variant parameters are optional. This is done by passing the parameter with the type set to `VT_ERROR` and a value of `DISP_E_PARAMNOTFOUND`. For example, if you want to do this with the *Var* parameter, the value it returns will depend on the object's implementation of the **IRequestDictionary** interface; the **QueryString** object will return the entire query string, for example.

- For **Request.QueryString**, *pVariantReturn* contains the unparsed query string data.
- For **Request.Form**, *pVariantReturn* contains the unparsed form data.
- For **Request.Cookies**, *pVariantReturn* contains a URL-encoded list of the cookies.
- **Request.ServerVariables** and **Response.Cookies** do not accept an optional *variant* parameter, and will raise a COM exception if *Var* is `DISP_E_PARAMNOTFOUND`.

If *Var* is not an optional parameter then it must be a `VT_BSTR` or a `VT_DISPATCH` pointer with a default value that can be converted to a `BSTR`. In this case, the `BSTR` value of *Var* is looked up in the appropriate dictionary, and the value of *Var* is returned. If *Var* is not in the dictionary, then a variant equal to `VT_EMPTY` is returned if the **IRequestDictionary** is covering one of the **Request** object's collections. If the **IRequestDictionary** pointer is the **Response.Cookies** collection, a new cookie with the name of *Var* is created, and that cookie is returned. If *Var* is not a `BSTR` (and not `DISP_E_PARAMNOTFOUND`), then the **get_Item** method will raise an OLE exception.

IRequestDictionary::get_Key

The **IRequestDictionary::get_Key** method returns the unique identifier for items in the **Cookies**, **Form** and **QueryString** collections.

```
HRESULT STDMETHODCALLTYPE get_Item(

  VARIANT VarKey,

  VARIANT * pVar
);
```

Parameters: IRequestDictionary::get_Key

VarKey

Identifier that indicates which item to retrieve from the collection.

pVar

Points to a variant that receives the item.

IRequestDictionary::get_NewEnum

The **IRequestDictionary::get_NewEnum** method returns an enumerator interface that can be used to iterate through the items in a collection.

```
HRESULT get_NewEnum(

  IUnknown ** ppEnumReturn

);
```

Parameters: IRequestDictionary::get_NewEnum

ppEnumReturn

Points to an **IUnknown** interface pointer that receives the enumerator.

Remarks: IRequestDictionary::get_NewEnum

You can use this method to iterate through the items in any collection. Members of the **Cookies** collection may be implemented as collections. **IReadCookie** and **IWriteCookie** both support this method, which you can use to iterate through members of the sub-collections.

IResponse Interface

The **IResponse** interface exposes the methods of the **Response** object.

IResponse::AddHeader	Adds a header to the HTML output.
IResponse::AppendToLog	Adds a string to the end of the Web server log for the current request.
IResponse::BinaryWrite	Writes data to the HTTP output without any character conversion.
IResponse::Clear	Erases any buffered HTML output.
IResponse::End	Causes the server to stop executing script and return the current response.
IResponse::Flush	Sends buffered HTML output immediately.
IResponse::get_Buffer	Retrieves the value of the **Buffer** property.
IResponse::get_CacheControl	Retrieves the value of the **CacheControl** property.
IResponse::get_CharSet	Retrieves the name of the character set to append to the content type header.
IResponse::get_ContentType	Retrieves the value of the **ContentType** property.

IResponse::get_Cookies	Retrieves the write-only **Cookies** collection.
IResponse::get_Expires	Retrieves the value of the **Expires** property.
IResponse::get_ExpiresAbsolute	Retrieves the value of the **ExpiresAbsolute** property.
IResponse::get_Status	Retrieves the value of the **Status** property.
IResponse::IsClientConnected	Determines if the client has disconnected from the server.
IResponse::PICS	Adds a value to the PICS label field of the **Response** header.
IResponse::put_Buffer	Sets the value of the **Buffer** property.
IResponse::putCharSet	Retrieves a character set to append to the content header type.
IResponse::put_ContentType	Sets the value of the **ContentType** property.
IResponse::put_Expires	Sets the value of the **Expires** property.
IResponse::put_ExpiresAbsolute	Sets the value of the **ExpiresAbsolute** property.
IResponse::put_Status	Sets the value of the **Status** property.
IResponse::Redirect	Causes the browser to attempt to connect to a different URL.
IResponse::Write	Writes a variant to the HTTP output.

The **IResponse** interface also supports the **IUnknown** and **IDispatch** interface methods.

IResponse::AddHeader

The **IResponse::AddHeader** method adds an HTTP header to the HTTP response.

```
HRESULT AddHeader(
  BSTR bstrHeaderName,
  BSTR bstrHeaderValue
);
```

Parameters: IResponse::AddHeader

bstrHeaderName

A binary string that contains the name of the HTTP header.

bstrHeaderValue

A binary string that contains the header value.

Remarks: IResponse::AddHeader

This method always adds an HTTP header with the specified value. It will not replace a header with the same name; once a header has been added it cannot be removed.

IResponse::AppendToLog

The **IResponse::AppendToLog** method adds a string to the end of the Web server log entry for the current request.

```
HRESULT AppendToLog(

  BSTR bstrLogEntry

);
```

Parameters: IResponse::AppendToLog

bstrLogEntry

A binary string to append to the log entry. The string may be a maximum of 80 characters and may not contain commas (,).

Remarks: IResponse::AppendToLog

This method adds a string to the end of the Web server log entry for this request. It can be called multiple times in one section of script; each time the method is called it appends the specified string to the existing entry.

IResponse::BinaryWrite

The IResponse::BinaryWrite method writes data to the HTTP output without converting Unicode characters to ANSI.

```
HRESULT BinaryWrite(

  VARIANT varBuffer

);
```

Parameters: IResponse::BinaryWrite

varBuffer

A variant containing the data as a Safe Array.

Remarks: IResponse::BinaryWrite

This method writes the specified information to the current HTTP output without any character conversion. This method is useful for writing binary data required by a custom application.

IResponse::Clear

The **IResponse::Clear** method erases any buffered HTML output.

```
HRESULT Clear(VOID)
```

Remarks: IResponse::Clear

This method only erases the response body; it does not erase response headers. You can use this method to handle error cases. This method will cause a run-time error if **Response.Buffer** has not been set to TRUE.

IResponse::End

The **IResponse::End** method causes the server to stop processing the script and to return the current response.

```
HRESULT End(VOID)
```

Remarks: IResponse::End

Any remaining contents of the file are not processed.

IResponse::Flush

The **IResponse::Flush** method sends buffered output immediately.

```
HRESULT Flush(VOID)
```

Remarks: IResponse::Flush

If this method is called on an ASP page, the server does not honor Keep-Alive requests for that page. This method will cause a run-time error if **Response.Buffer** has not been set to TRUE.

IResponse::get_Buffer

The **IResponse::get_Buffer** method retrieves the current value of the **Buffer** property of the **Response** object.

```
HRESULT get_Buffer(

  VARIANT_BOOL * fIsBuffering
```

);

Parameters: IResponse::get_Buffer

fIsBuffering

Points to a Boolean variant that receives the **Buffer** value.

Remarks: IResponse::get_Buffer

When page output is buffered, the server does not send a response to the client until all of the server scripts on the current page have been processed, or until the **Flush** or **End** method has been called.

The **Buffer** property cannot be set after the server has sent output to the client. For this reason, the call to **Response.Buffer** should be the first line of the .asp file.

IResponse::get_CacheControl

The **IResponse::get_CacheControl** method retrieves a value for the **CacheControl** property.

```
HRESULT get_CacheControl(

  BSTR * pbstrCacheControl

);
```

Parameters: IResponse::get_CacheControl

pbstrCacheControl

Points to a binary string that receives the **CacheControl** value.

Remarks: IResponse::get_CacheControl

You can use the **CacheControl** value to override the default value (FALSE). By setting the value to TRUE, proxy servers will be able to cache output from ASP pages.

IResponse::get_CharSet

The **IResponse::get_CharSet** method retrieves a character set to append to the content type header.

```
HRESULT get_CharSet(

  BSTR * pbstrCharSetRet

);
```

Parameters: IResponse::get_CharSet

pbstrCharSetRet

Points to a binary string that receives the **CharSet** value.

Remarks: IResponse::get_CharSet

You can use the **CharSet** value to determine the character set to use when displaying the current **Response** object.

IResponse::get_ContentType

The **IResponse::get_ContentType** method retrieves the current value of the **ContentType** property of the **Response** object.

```
HRESULT get_ContentType(

 BSTR * pbstrContentTypeRet
```

);

Parameters: IResponse::get_ContentType

pbstrContentTypeRet

Points to a binary string that receives the **ContentType** value.

Remarks: IResponse::get_ContentType

You can use the *pbstrContentTypeRet* value to determine the content type of the current **Response** object.

IResponse::get_Cookies

The **IResponse::get_Cookies** method retrieves the write-only **Cookies** collection of the **Response** object.

```
HRESULT get_Cookies(

  IRequestDictionary ** ppCookies
```

);

Parameters: IResponse::get_Cookies

ppCookies

Points to an **IRequestDictionary** interface pointer that receives the write-only **Cookies** collection.

Remarks: IResponse::get_Cookies

You can iterate through the **Cookies** collection with the **IRequestDictionary::get_NewEnum** method, or you can retrieve a specific cookie with the **IRequestDictionary::get_Item** method.

IResponse::get_Expires

The **IResponse::get_Expires** method retrieves the current value of the **Expires** property of the **Response** object.

```
HRESULT get_Expires*

  VARIANT * pvarExpiresMinutesRet
```

);

Parameters: IResponse::get_Expires

pvarExpiresMinutesRet

Points to a variant that receives the **Expires** value. This value specifies the number of minutes until the page will expire.

Remarks: IResponse::get_Expires

If the user returns to the same page before it expires, the cached version is displayed. If this property is set more than once on a page, the shortest time is used.

IResponse::get_ExpiresAbsolute

The **IResponse::get_ExpiresAbsolute** method retrieves the current value of the **ExpiresAbsolute** property of the **Response** object.

```
HRESULT get_ExpiresAbsolute(

  VARIANT * pvarExpiresRet
);
```

Parameters: IResponse::get_ExpiresAbsolute

pvarExpiresRet

Points to a variant that receives the **ExpiresAbsolute** value. The variant should specify the date and time.

Remarks: IResponse::get_ExpiresAbsolute

If the user returns to the same page before the set date and time, the cached version is displayed. If a time is not specified, the page expires at midnight of that day. If a date is not specified, the page expires at the given time on the day that the script is run. If this property is set more than once on a page, the earliest expiration date or time is used. The date value sent in the Expires header conforms to the RFC-1123 date format. The time value is converted to Greenwich Mean Time before an Expires header is sent.

IResponse::get_Status

The **IResponse::get_Status** method retrieves the current value of the **Status** property of the **Response** object.

```
HRESULT get_Status(

  BSTR * pbstrStatusRet
);
```

Parameters: IResponse::get_Status

pbstrStatusRet

Points to a binary string that receives the **Status** value.

Remarks: IResponse::get_Status

Use this property to modify the status line returned by the server. Status values are defined in the HTTP specification.

IResponse::IsClientConnected

The **IResponse::IsClientConnected** method determines if the client has disconnected from the server since the last **Response.Write** operation.

```
HRESULT IsClientConnected(

  VARIANT_BOOL pfIsClientConnected

);
```

Parameters: IResponse::IsClientConnected

pfIsClientConnected

Points to a Boolean that indicates whether or not the client is connected.

IResponse::PICS

The **IResponse::PICS** methods adds a value to the PICS label field of the response header.

```
HRESULT Pics(

  BSTR bstrHeaderValue

);
```

Parameters: IResponse::PICS

bstrHeaderValue

A binary string that contains the new **PICS** value.

Remarks: IResponse::PICS

The **IResponse::PICS** method inserts any string in the header, whether or not it represents a valid PICS label.

If a single page contains multiple tags containing **Response.PICS**, each instance will replace the PICS label set by the previous one. As a result, the PICS label will be set to the value specified by the last instance of **Response.PICS** in the page.

Because PICS labels contain quotes, the author must replace each quote with the ASCII equivalent for the quote symbol [& chr(34) &].

For more details on the PICS standard, see:

http://www.w3.org/PICS/

IResponse::put_Buffer

The **IResponse::put_Buffer** method sets the value of the **Buffer** property of the **Response** object.

```
HRESULT put_Buffer(

  VARIANT_BOOL fIsBuffering
);
```

Parameters: IResponse::put_Buffer

fIsBuffering

A Boolean variant that contains the new **Buffer** value.

Remarks: IResponse::put_Buffer

When page output is buffered, the server does not send a response to the client until all of the server scripts on the current page have been processed, or until the **Flush** or **End** methods are called.

Since the **Buffer** property cannot be set after the server has sent output to the client, the call to **Response.Buffer** should be the first line of the .asp file.

IResponse::putCharSet

The **IResponse::putCharSet** method sets the value of the **Charset** property of the **Response** object.

```
HRESULT put_CharSet(

  BSTR bstrCharset
);
```

Parameters: IResponse::putCharSet

bstrCharSet

A binary string that contains the new **CharSet** value.

Remarks: IResponse::putCharSet

You can use the **CharSet** value to set the character set to use when displaying the **Response** object.

IResponse::put_ContentType

The **IResponse::put_ContentType** method sets the value of the **ContentType** property of the **Response** object.

```
HRESULT put_ContentType(
```

```
    BSTR bstrContentType
);
```

Parameters: IResponse::put_ContentType

bstrContentType

A binary string that contains the new **ContentType** value.

Remarks: IResponse::put_ContentType

You can use the **ContentType** value to set the content type to use when displaying the current **Response** object.

IResponse::put_Expires

The **IResponse::put_Expires** method sets the current value of the **Expires** property of the **Response** object.

```
HRESULT put_Expires(

 LONG lExpiresMinutes  //LONG that contains the new Expires value
);
```

Parameters: IResponse::put_Expires

lExpiresMinutes

A Long integer that contains the new **Expires** value.

Remarks: IResponse::put_Expires

If the user returns to the same page before it expires, the cached version is displayed. If this method is set more than once on a page, the shortest time is used.

IResponse::put_ExpiresAbsolute

The **IResponse::put_ExpiresAbsolute** method sets the value of the **ExpiresAbsolute** property of the **Response** object.

```
HRESULT put_ExpiresAbsolute(

 DATE dtExpires
);
```

Parameters: IResponse::put_ExpiresAbsolute

dtExpires

A date that contains the new **ExpiresAbsolute** value

Remarks: IResponse::put_ExpiresAbsolute

If the user returns to the same page before the set date and time, the cached version is displayed. If a time is not specified, the page expires at midnight of that day. If a date is not specified, the page expires at the given time on the day that the script is run. If this property is set more than once on a page, the earliest expiration date or time is used. The date value sent in the Expires header conforms to the RFC-1123 date format. The time value is converted to Greenwich Mean Time before an Expires header is sent.

IResponse::put_Status

The **IResponse::put_Status** method sets the value of the **Status** property of the **Response** object.

```
HRESULT put_Status(

 BSTR bstrStatus

);
```

Parameters: IResponse::put_Status

bstrStatus

A binary string that contains the new **Status** value.

Remarks: IResponse::put_Status

You can use this property to modify the status line returned by the server. Status values are defined in the HTTP specification.

IResponse::Redirect

The **IResponse::Redirect** method first stops the server from processing the current script and then causes the browser to attempt to connect to a different URL. For more information, see the **Redirect** method of the **Response** object.

```
HRESULT Redirect(

 BSTR bstrURL

);
```

Parameters: IResponse::Redirect

bstrURL

A binary string that contains the URL.

Remarks: IResponse::Redirect

If you have set any response body content in the page, it will be ignored. However, this method does send to the client other HTTP headers set by this page. An automatic response body

containing the redirect URL as a link is generated. The **IResponse::Redirect** method sends the following explicit header,

```
HTTP/1.0 302 Object Moved

Location URL
```

where `URL` is the value passed to the method.

IResponse::Write

The **IResponse::Write** method writes the specified variant to the HTTP output. For more information, see the **Write** method of the **Response** object.

```
HRESULT Write(

 VARIANT varText

);
```

Parameters: IResponse::Write

varText

A variant to write to the HTTP output.

Remarks: IResponse::Write

If VBScript is your primary scripting language, *variant* cannot be a string literal that contains more than 1022 characters. This is because VBScript limits static strings to 1022 bytes. You can, however, specify *variant* as the name of a variable that contains greater than 1022 bytes.

IScriptingContext Interface

The **IScriptingContext** interface exposes methods that your component can use to retrieve the ASP built-in objects.

IScriptingContext::get_Application	Retrieves the **Application** object. This object implements the **IApplicationObject** interface.
IScriptingContext::get_Request	Retrieves the **Request** object. This object implements the **IRequest** interface.
IScriptingContext::get_Response	Retrieves the **Response** object. This object implements the **IResponse** interface.
IScriptingContext::get_Server	Retrieves the **Server** object. This object implements the **IServer** interface.
IScriptingContext::get_Session	Retrieves the **Session** object. This object implements the **ISessionObject** interface.

The **IScriptingContext** also supports the **IUnknown** and **IDispatch** interface methods.

IScriptingContext::get_Application

The **IScriptingContext::get_Application** method retrieves a pointer to the **IApplicationObject** interface of the **Application** object.

```
HRESULT get_Application(

 IApplicationObject ** ppApplication
```

);

Parameters: IScriptingContext::get_Application

ppApplication

Points to an **IApplicationObject** interface.

Remarks: IScriptingContext::get_Application

You can use **IApplicationObject** interface to gain access to variables and objects that have been given application scope. **IApplicationObject** also exposes methods to lock and unlock an application to prevent concurrent access to application objects and variables.

IScriptingContext::get_Request

The **IScriptingContext::get_Request** method retrieves a pointer to the **IRequest** interface of the **Request** object.

```
HRESULT get_Request(

 IRequest ** ppRequest
```

);

Parameters: IScriptingContext::get_Request

ppRequest

Points to an **IRequest** interface pointer.

Remarks: IScriptingContext::get_Request

You can use the **IRequest** interface to access the methods, properties, and collections of the **Request** object.

IScriptingContext::get_Response

The **IScriptingContext::get_Response** method retrieves a pointer to the **IResponse** interface of the **Response** object.

```
HRESULT get_Response(
```

```
    IResponse ** ppResponse
);
```

Parameters: IScriptingContext::get_Response

ppResponse

Points to an **IResponse** interface pointer.

Remarks: IScriptingContext::get_Response

You can use the **IResponse** interface to access the methods, properties, and collections of the **Response** object.

IScriptingContext::get_Server

The **IScriptingContext::get_Server** method retrieves a pointer to the **IServer** interface of the **Server** object.

```
HRESULT get_Server(

 IServer ** ppServer
);
```

Parameters: IScriptingContext::get_Server

ppServer

Points to an **IServer** interface pointer.

Remarks: IScriptingContext::get_Server

You can use the **IServer** interface to access the methods exposed by the **Server** object.

IScriptingContext::get_Session

The **IScriptingContext::get_Session** method retrieves a pointer to the **ISessionObject** interface of the **Session** object.

```
HRESULT get_Session(

 ISessionObject ** ppSession
);
```

Parameters: IScriptingContext::get_Session

ppSession

Points to an **ISessionObject** interface pointer.

Remarks: IScriptingContext::get_Session

You can use the **ISessionObject** interface to access variables and objects that have been given session scope.

IServer Interface

The **IServer** interface exposes the methods and properties of the **Server** object.

IServer::CreateObject	Creates an instance of an object.
IServer::get_ScriptTimeout	Retrieves the value of the **ScriptTimeout** property.
IServer::HTMLEncode	Applies HTML encoding to the specified string.
IServer::MapPath	Maps the specified relative or virtual path to the corresponding physical directory on the server.
IServer::put_ScriptTimeout	Sets the value of the **ScriptTimeout** property.
IServer::URLEncode	Applies URL encoding rules, including escape characters, to the specified string.

The **IServer** interface also supports the **IUnknown** and **IDispatch** interface methods.

IServer::CreateObject

The **IServer::CreateObject** method creates an instance of a server component. For more information, see the **CreateObject** method of the **Server** object.

```
HRESULT CreateObject(
 BSTR bstrProgID,
 IDispatch ** ppDispObject
);
```

Parameters: IServer::CreateObject

bstrProgID

A binary string that contains the progID of the object.

ppDispObject

Points to an **IDispatch** interface pointer.

Remarks: IServer::CreateObject

By default, objects created by the **Server.CreateObject** method have page scope. This means that they are automatically destroyed by the server when it finishes processing the current ASP page.

To create an object with session or application scope, you can either use the <OBJECT> tag and set the SCOPE attribute to SESSION or APPLICATION, or store the object in a session or application variable.

IServer::get_ScriptTimeout

The **IServer::get_ScriptTimeout** method retrieves the value of the **ScriptTimeout** property of the **Server** object.

```
HRESULT get_ScriptTimeout(

 LONG * plTimeoutSeconds
);
```

Parameters: IServer::get_ScriptTimeout

plTimeoutSeconds

Points to a LONG that receives the **ScriptTimeout** value.

Remarks: IServer::get_ScriptTimeout

A default **ScriptTimeout** can be set for a Web Service or Web Server by using the **ScriptTimeout** property in the registry or configuration file. The **ScriptTimeout** property cannot be set to a value less than that specified in the registry or configuration file.

IServer::HTMLEncode

The **IServer::HTMLEncode** method applies HTML encoding to the specified string. For more information, see the **HTMLEncode** method of the **Server** object.

```
HRESULT HTMLEncode(

 BSTR bstrIn,

 BSTR * pbstrEncoded
);
```

Parameters: IServer::HTMLEncode

bstrIn
A binary string that contains the text to be encoded.

pbstrEncoded

Points to a binary string that receives the encoded text.

Remarks: IServer::HTMLEncode

If your component returns the encoded text to a browser, the browser will display it in HTML format, rather than in plain text. For example, if the bstrIn contained the following string, < >,

the `pbstrEncoded` parameter would contain the HTML code for those characters, `&lt &gt`. If your component returned this to a browser, however, it would display it as < >.

IServer::MapPath

The **IServer::MapPath** method maps the specified relative or virtual path to the corresponding physical directory on the server. For more information, see the **MapPath** method of the **Server** object.

```
HRESULT MapPath(
 BSTR bstrLogicalPath,
 BSTR * pbstrPhysicalPath
);
```

Parameters: IServer::MapPath

bstrLogicalPath

A binary string that contains the relative or virtual path.

pbstrPhysicalPath

Points to a binary string that receives the physical path.

Remarks: IServer::MapPath

This method does not check whether the path it returns is valid or exists on the server.

Because the **IServer::MapPath** method maps a path regardless of whether the specified directories currently exist, you can use it to map a path to a physical directory structure, and then pass that path to a component that creates the specified directory or file on the server.

IServer::put_ScriptTimeout

The **IServer::put_ScriptTimeout** method sets the value of the **ScriptTimeout** property of the **Server** object.

```
HRESULT put_ScriptTimeout(
 LONG lTimeoutSeconds
);
```

Parameters: IServer::put_ScriptTimeout

lTimeoutSeconds

A LONG integer that contains the new **ScriptTimeout** value.

Remarks: IServer::put_ScriptTimeout

A default **ScriptTimeout** can be set for a Web Service or Web Server by using the **ScriptTimeout** property in the registry or configuration file. The **ScriptTimeout** property cannot be set to a value less than that specified in the registry or configuration file.

IServer::URLEncode

The **IServer::URLEncode** method applies URL encoding rules, including escape characters, to the specified string. For more information, see the **URLEncode** method of the **Server** object.

```
HRESULT URLEncode(
 BSTR bstrIn,
 BSTR * pbstrEncoded
);
```

Parameters: IServer::URLEncode

bstrIn

A binary string that contains the text to be encoded.

pbstrEncoded

Points to a binary string that receives the encoded text.

ISessionObject Interface

The **ISessionObject** interface exposes the methods of the **Session** object.

ISessionObject::Abandon	Destroys all the objects stored in the **Session** object and releases their resources.
ISessionObject::get_Contents	Retrieves the **Contents** collection.
ISessionObject::get_SessionID	Retrieves the value of the **SessionID** entry.
ISessionObject::get_StaticObjects	Retrieves the **StaticObjects** collection.
ISessionObject::get_Timeout	Retrieves the value of the **Timeout** property.
ISessionObject::get_Value	Retrieves the value of a variable stored in the **Session** object.
ISessionObject::put_Timeout	Sets a value for the **Timeout** property.
ISessionObject::putref_Value	Stores a variable in the **Session** object by reference.
ISessionObject::put_Value	Stores a variable in the **Session** object by value.

The **ISessionObject** interface also supports the **IUnknown** and **IDispatch** interfaces.

ISessionObject::Abandon

The **ISessionObject::Abandon** method destroys all the objects stored in the current session and releases their resources. For more information see the **Abandon** method of the **Session** object.

```
HRESULT Abandon(VOID);
```

Remarks: ISessionObject::Abandon

When this method is called, the current **Session** object is queued for deletion, but is not actually deleted until all of the script commands on the current page have been processed. This means that you can access variables stored in the **Session** object on the same page as the call to **Abandon**, but not in any subsequent Web pages.

ISessionObject::get_Contents

The **ISessionObject::get_Contents** method retrieves the value of the **Contents** property.

```
HRESULT get_Contents(

 IVariantDictionary **ppProperties

);
```

Parameters: ISessionObject::get_Contents

ppProperties

Points to an **IVariantDictionary** interface pointer that receives the **Contents** collection.

Remarks: ISessionObject::get_Contents

The **Session Contents** collection contains all variables and objects that have been given application scope with the **Server.CreateObject** command. You can iterate through the **Contents** collection with the **IVariantDictionary::get_NewEnum** method exposed by the interface. You can also retrieve a specific member of the collection with the **IVariantDictionary::get_Item** method.

ISessionObject::get_SessionID

The **ISessionObject::get_SessionID** method retrieves the value of the **SessionID** property.

```
HRESULT get_SessionID(

 BSTR * pbstrRet

);
```

Parameters: ISessionObject::get_SessionID

pbstrRet

Points to a binary string that receives the **SessionID** value.

Remarks: ISessionObject::get_SessionID

You should not use the **SessionID** property to generate primary key values for a database application. This is because if the Web server is restarted, some **SessionID** values may be the same as those generated before the server was stopped. Instead, you should use an auto-increment column data type.

ISessionObject::get_StaticObjects

The **ISessionObject::get_StaticObjects** method retrieves the **StaticObjects** collection of the **Session** object.

```
HRESULT get_StaticObjects(

 IVariantDictionary **ppProperties
);
```

Parameters: ISessionObject::get_StaticObjects

ppProperties

Points to an **IVariantDictionary** interface pointer that receives the **StaticObjects** collection.

Remarks: ISessionObject::get_StaticObjects

You can iterate through the **StaticObjects** collection with the **IVariantDictionary::get_NewEnum** method exposed by the interface. You can also retrieve a specific member of the collection with the **IVariantDictionary::get_Item** method.

ISessionObject::get_Timeout

The **ISessionObject::get_Timeout** method retrieves the value of the **Timeout** property of the **Session** object.

```
HRESULT get_Timeout(

 LONG * plvar
);
```

Parameters: ISessionObject::get_Timeout

plvar

Points to a LONG integer that receives the **Timeout** value.

ISessionObject::get_Value

The **ISessionObject::get_Value** method retrieves the value of a variable stored in the **Session** object.

```
HRESULT get_Value(
 BSTR bstrValue,
 VARIANT * pvar
);
```

Parameters: ISessionObject::get_Value

bstrValue

A binary string that contains the variable name.

pvar

Points to a variant that receives the variable value.

Remarks: ISessionObject::get_Value

You can store values in the **Session** object. Information stored in the **Session** object is available throughout the session and has session scope.

ISessionObject::put_Timeout

This **ISessionObject::put_Timeout** method sets the value of the **Timeout** property of the **Session** object.

```
HRESULT put_Timeout(
 LONG lvar
);
```

Parameters: ISessionObject::put_Timeout

lvar

A LONG integer that contains the new **Timeout** value.

ISessionObject::putref_Value

The **ISessionObject::putref_Value** method stores a COM object in the **Session** object. This method is equivalent to **ISessionObject::put_Value** for non-COM objects.

```
HRESULT putref_Value(
 BSTR bstrValue,
VARIANT var
);
```

Parameters: ISessionObject::putref_Value

bstrValue

A binary string that contains the new variable name.

var

A variant to store in the variable.

Remarks: ISessionObject::putref_Value

Before you store an object in the **Session** object, you should know what threading model it uses. For more information, see the Scope and Threading topics.

ISessionObject::put_Value

The **ISessionObject::put_Value** method stores a variant in the **Session** object. If the variant is a COM object, then Sun Chili!Soft ASP will attempt to store the default value of that object into the application. If Sun Chili!Soft ASP cannot get the default value, then the call will fail.

```
HRESULT put_Value(
 BSTR bstrValue,
 VARIANT var
);
```

Parameters: ISessionObject::put_Value

bstrValue

A binary string that contains the variable name.

var

A variant that contains the variable value.

Remarks: ISessionObject::put_Value

If you store an array in a **Session** object, you should not attempt to alter the elements of the stored array directly. For example, the following script will not work.

```
<% Session("StoredArray")(3) = "new value" %>
```

This is because the **Session** object is implemented as a collection. The array element `StoredArray(3)` does not receive the new value. Instead, the value is indexed into the collection, overwriting any information stored at that location.

It is strongly recommended that if you store an array in the **Session** object, you retrieve a copy of the array before retrieving or changing any of the elements of the array. When you are done with the array, you should store the array in the **Session** object again so that any changes you made are saved.

IStringList Interface

The **IStringList** interface is used to retrieve individual items from the string lists contained in the **QueryString**, **Form**, or **ServerVariables** collections.

IStringList::get_Count	Retrieves the number of items in the string list.
IStringList::get_Item	Retrieves the specified item from a string list.
IStringList::get_NewEnum	Retrieves an enumerator for the string list.

The **IStringList** interface also supports the **IUnknown** and **IDispatch** interface methods.

IStringList::get_Count

The **IStringList::get_Count** method retrieves the number of items in a string list.

```
HRESULT get_Count(

 INT * cStrRet

);
```

Parameters: IStringList::get_Count

cStrRet

Points to an integer that receives the number of items.

Remarks: IStringList::get_Count

The **Form**, **QueryString**, and **ServerVariables** collections return parameter values as an array. If the collection does not contain multiple values, the `cStrRet` will contain 1; if the parameter is not found, it will contain 0.

You can use this method to determine the number of items in the array. For example, if a **Form** included an *Items ordered* parameter, you could use the **get_Count** method to determine the number of items that had been ordered in the **Form**.

IStringList::get_Item

The **IStringList::get_Item** method retrieves an individual item from a string list.

```
HRESULT get_Item(

 VARIANT Var,

 VARIANT * pVariantReturn

);
```

Parameters: IStringList::get_Item

Var

A variant that contains the name of the item in the collection.

pVariantReturn

Points to a variant that receives the item value.

Remarks: IStringList::get_Item

You can use this method to retrieve a particular item from an array that has been returned by a **Form**, **QueryString**, or **ServerVariables** collection.

IStringList::get_NewEnum

The **IStringList::get_NewEnum** method retrieves an enumerator interface which can be used to iterate through the items in the string list.

```
HRESULT get__NewEnum(

 IUnknown ** ppEnumReturn
```

);

Parameters: IStringList::get_NewEnum

ppEnumReturn

Points to an **IUnknown** interface pointer that receives the enumerator.

Remarks: IStringList::get_NewEnum

The **Form**, **QueryString**, and **ServerVariables** collections return parameter values as an array. You can use this method to retrieve an enumerator to iterate through one of the string list collections.

IVariantDictionary Interface

The **IVariantDictionary** interface exposes methods that you use to access the members of the **Application Contents**, **Application StaticObjects**, **Session Contents** and **Session StaticObjects** collections.

IVariantDictionary::get_Count	Retrieves the number of items in the collection.
IVariantDictionary::get_Item	Retrieves an item from the specified collection.
IVariantDictionary::get_Key	Retrieves an identifier for an item.
IVariantDictionary::get_NewEnum	Retrieves an enumerator for the collection.
IVariantDictionary::put_Item	Adds an item to the specified collection.

IVariantDictionary::get_Count

The **IVariantDictionary::get_Count** method returns the number of items in either the **Contents** or **StaticObjects** collection.

```
HRESULT STDMETHODCALLTYPE get_Count(

  int * cStrRet

);
```

Parameters: IVariantDictionary::get_Count

cStrRet

Points to an integer that contains a total of the number of items in the collection.

IVariantDictionary::get_Item

The **IVariantDictionary::get_Item** method retrieves the specified item from either the **Contents** or **StaticObjects** collection. Both the **Application** and **Session** objects provide these collections.

```
HRESULT STDMETHODCALLTYPE get_Item(

  VARIANT VarKey,

  VARIANT * pvar

);
```

Parameters: IVariantDictionary::get_Item

VarKey

Identifier that indicates which item to retrieve from the collection.

pVar

Points to the item that is returned.

IVariantDictionary::get_Key

The **IVariantDictionary::get_Key** method returns a unique identifier for an item in either the **Contents** or **StaticObjects** collection.

```
HRESULT STDMETHODCALLTYPE get_Item(

  VARIANT VarKey,

  VARIANT * pvar

);
```

Parameters: IVariantDictionary::get_Key

VarKey

Identifier that indicates which item to retrieve from the collection.

pVar

Points to the item that is returned.

IVariantDictionary::get_NewEnum

The **IVariantDictionary::get_NewEnum** method retrieves an enumerator interface which can be used to iterate through the items in the collection.

```
HRESULT STDMETHODCALLTYPE get__NewEnum(

  IUnknown * ppEnumReturn
);
```

Parameters: IVariantDictionary::get_NewEnum

ppEnumReturn

Points to an **IUnknown** interface pointer that receives the enumerator.

IVariantDictionary::put_Item

The **IVariantDictionary::put_Item** method adds an item to either the **Contents** or **StaticObjects** collection.

```
HRESULT put_Item(

  VARIANT VarKey,

  VARIANT var
);
```

Parameters: IVariantDictionary::put_Item

VarKey

Identifier that indicates which item to add to the collection.

Var

The item to add to the collection.

IWriteCookie Interface

The **IWriteCookie** interface exposes the methods that you can use to alter the values and attributes of the cookies stored in the write-only **Cookies** collection.

IWriteCookie::get_NewEnum	Retrieves an enumerator for the **Cookies** collection.
IWriteCookie::get_HasKeys	Retrieves a Boolean indicating whether the cookie has keys.
IWriteCookie::put_Domain	Sets the **Domain** attribute of the cookie to the specified value.
IWriteCookie::put_Expires	Sets the **Expires** attribute of the cookie to the specified value.
IWriteCookie::put_Item	Adds an item to the **Cookies** collection.
IWriteCookie::put_Path	Sets the **Path** attribute of the cookie to the specified value.
IWriteCookie::put_Secure	Sets the **Secure** attribute of the cookie to the specified value.

IWriteCookie and **IReadCookie** are interfaces for the same object. If you have an **IWriteCookie** pointer, you can call the **IUnknown::QueryInterface** method on an **IReadCookie** pointer.

The **IWriteCookie** interface also supports the **IUnknown** and **IDispatch** interface methods.

IWriteCookie::get_NewEnum

The **IWriteCookie::get_NewEnum** method retrieves an enumerator interface which can be used to iterate through the items in a cookie.

```
HRESULT get__NewEnum(

 IUnknown ** ppEnumReturn
```

);

Parameters: IWriteCookie::get_NewEnum

ppEnumReturn

Points to an **IUnknown** interface pointer that receives the enumerator.

Remarks: IWriteCookie::get_NewEnum

If a cookie has been implemented as a collection, you can use this method to iterate through all the members of its collection.

IWriteCookie::get_HasKeys

The **IWriteCookie::get_HasKeys** method retrieves a Boolean value which indicates whether or not the cookie has keys.

```
HRESULT get_HasKeys(
```

```
    VARIANT_BOOL * pfHasKeys
);
```

Parameters: IWriteCookie::get_HasKeys

pfHasKeys

Points to a Boolean that indicates whether or not the cookie has keys.

Remarks: IWriteCookie::get_HasKeys

If the cookie has been implemented as a collection, this method will return TRUE. If the cookie has keys, you can use the IWriteCookie::get_NewEnum method to retrieve an enumerator for iterating through the collection.

IWriteCookie::put_Domain

The **IWriteCookie::put_Domain** method sets the **Domain** attribute of the write-only **Cookies** collection to the specified value.

```
HRESULT put_Domain(

 BSTR bstrDomain
);
```

Parameters: IWriteCookie::put_Domain

bstrDomain

A binary string that contains the new domain value.

Remarks: IWriteCookie::put_Domain

You can use this method to specify particular domains for a cookie. This method only applies to the **Response.Cookies** collection.

IWriteCookie::put_Expires

The **IWriteCookie::put_Expires** method sets the **Expires** attribute of the write-only **Cookies** collection to the specified value.

```
HRESULT put_Expires(

 DATE dtExpires
);
```

Parameters: IWriteCookie::put_Expires

dtExpires

A date that contains the new expiration value.

Remarks: IWriteCookie::put_Expires

This date must be set in order for the cookie to be maintained after the session ends. If this attribute is not set to a date beyond the current date, the cookie will expire when the session ends.

IWriteCookie::put_Item

The **IWriteCookie::put_Item** method adds a specified cookie to the write-only **Cookie** collection.

```
HRESULT put_Item(

 VARIANT key,

 BSTR bstrValue
);
```

Parameters: IWriteCookie::put_Item

key

A variant that contains the name of the cookie.

bstrValue

A binary string that contains the cookie value.

Remarks: IWriteCookie::put_Item

Automation objects can specify that one or more variant parameters are optional. This is done by passing the parameter with the type set to `VT_ERROR` and a value of `DISP_E_PARAMNOTFOUND`. If you pass *key* as an optional parameter, the cookie is treated as a simple cookie and its value is set to *bstrValue.* Otherwise, the cookie is treated as a dictionary cookie and *bstrValue* is the value for the cookie's key.

IWriteCookie::put_Path

The **IWriteCookie::put_Path** method sets the **Path** attribute of the write-only **Cookies** collection to the specified value.

```
HRESULT put_Path(

BSTR bstrPath
);
```

Parameters: IWriteCookie::put_Path

bstrPath

A binary string that contains the new path.

Remarks: IWriteCookie::put_Path

You can use this method to specify that the cookie should be sent only to requests on a particular path. If this attribute is not set, the application path is used.

IWriteCookie::put_Secure

The **IWriteCookie::put_Secure** method sets the **Secure** attribute of the write-only **Cookies** collection to the specified value.

```
HRESULT put_Secure(

 VARIANT_BOOL fSecure
);
```

Parameters: IWriteCookie::put_Secure

fSecure

A Boolean that contains the new value for the **Secure** attribute.

Remarks: IWriteCookie::put_Secure

You can use this method to specify that a cookie is secure. If you set this value to TRUE, a `Secure` flag will be added to the `Set-Cookie` header sent to the client. The `Secure` flag instructs the client to use only secure means to access the server when sending back the cookie.

JScript Language Reference

Sun Chili!Soft ASP supports version 3.1 of JScript.

This section provides reference information on the following JScript topics:

- JScript Operators
- JScript Statements
- JScript Functions
- JScript Objects
- JScript FileSystemObject Collections

JScript Features (ECMA)

Category	Feature/Keyword
Array Handling	JScript Array Object
	JScript Array Object concat Method

	JScript Array Object join Method
	JScript Array Object length Property
	JScript Array Object reverse Method
	JScript Array Object slice Method
	JScript Array Object sort Method
Assignments	JScript Assignment Operator (=) [JScript Assignment Operator ('equals sign')
	JScript Compound Assignment Operators
Booleans	JScript Boolean Object
Comments	JScript Comment Statements
Constants/Literals	JScript Global Object NaN Property
	null,
	true,
	false,
	JScript Global Object Infinity Property,
	Undefined
Control Flow	JScript break Statement
	JScript continue Statement
	JScript for Statement
	JScript for. . . in Statement
	JScript if. . . else Statement
	JScript return Statement
	JScript while Statement
Dates and Time	JScript Date Object
	JScript Date Object getDate Method
	JScript Date Object getDay Method
	JScript Date Object getFullYear Method
	JScript Date Object getHours Method
	JScript Date Object getMilliseconds Method
	JScript Date Object getMinutes Method
	JScript Date Object getMonth Method
	JScript Date Object getSeconds Method

Declarations	JScript function Statement
	JScript new Operator
	JScript this Statement
	JScript var Statement
	JScript with Statement
Function Creation	JScript Function Object
	JScript Function Object arguments Property
	JScript Function Object length Property
Global Methods	JScript Global Object
	JScript Global Object escape Method
	JScript Global Object unescape Method
	JScript Global Object eval Method
	JScript Global Object isFinite Method
	JScript Global Object isNaN Method
	JScript Global Object parseInt Method
	JScript Global Object parseFloat Method
Math	JScript Math Object
	JScript Math Object abs Method
	JScript Math Object acos Method
	JScript Math Object asin Method
	JScript Math Object atan Method
	JScript Math Object atan2 Method
	JScript Math Object ceil Method
	JScript Math Object cos Method
	JScript Math Object exp Method
	JScript Math Object floor Method
	JScript Math Object log Method
	JScript Math Object max Method
	JScript Math Object min Method
	JScript Math Object pow Method
	JScript Math Object random Method
	JScript Math Object round Method

	JScript Math Object sin Method
	JScript Math Object sqrt Method
	JScript Math Object tan Method
	JScript Math Object E Property
	JScript Math Object LN2 Property
	JScript Math Object LN10 Property
	JScript Math Object LOG2E Property
	JScript Math Object LOG10E Property
	JScript Math Object PI Property
	JScript Math Object SQRT1_2 Property
	JScript Math Object SQRT2 Property
Numbers	JScript Number Object
	JScript Number Object MAX_VALUE Property
	JScript Number Object MIN_VALUE Property
	JScript Number Object NaN Property
	JScript Number Object NEGATIVE_INFINITY Property
	JScript Number Object POSITIVE_INFINITY Property
Object Creation	JScript Object
	JScript new Operator
	JScript Object constructor Property
	JScript Object prototype Property
	JScript Object toString Method
	JScript Object valueOf Method
Operators	JScript Addition Operator (+)
	JScript Subtraction and Unary Negation Operator (-)
	JScript Modulus Operator (%)
	JScript Multiplication Operator (*)
	JScript Division Operator (/)
	JScript Subtraction and Unary Negation Operator (-)
	JScript Comparison Operators
	JScript Logical AND Operator (&&)
	JScript Logical OR Operator (||)

	JScript String Object toUpperCase Method
	JScript String Object length Property

JScript Features (non-ECMA)

Category	Feature/Keyword
Array Handling	JScript Array Object concat Method
	JScript Array Object slice Method
	JScript VBArray Object
	JScript VBArray Object Dimensions Method
	JScript VBArray Object getItem Method
	JScript VBArray Object lbound Method
	JScript VBArray Object toArray Method
	JScript VBArray Object ubound Method
Conditional Compilation	JScript @cc_on Statement
	JScript @if Statement
	JScript @set Statement
	JScript Conditional Compilation Variables
Control Flow	JScript do. . . while Statement
	JScript Labeled Statement
	JScript switch Statement
Dates and Time	JScript Date Object getVarDate Method
Enumeration	JScript Enumerator Object
	JScript Enumerator Object AtEnd Method
	JScript Enumerator Object item Method
	JScript Enumerator Object moveFirst Method
	JScript Enumerator Object moveNext Method
Function Creation	JScript Function Object caller Property
Operators	JScript Comparison Operators
Objects	JScript Enumerator Object
	JScript RegExp Object
	JScript Regular Expression Object

	JScript VBArray Object
Regular Expressions and Pattern Matching	JScript RegExp Object
	JScript RegExp Object index Property
	JScript RegExp Object input Property
	JScript RegExp Object lastIndex Property
	JScript RegExp Object lastMatch Property
	JScript RegExp Object lastParen Property
	JScript RegExp Object leftContext Property
	JScript RegExp Object multiline Property
	JScript RegExp Object rightContext Property
	JScript RegExp Object $1. . . $9 Property
	JScript Regular Expression Object
	JScript Regular Expression Object global Property
	JScript Regular Expression Object ignoreCase Property
	JScript Regular Expression Object lastIndex Property
	JScript Regular Expression Object source Property
	JScript Regular Expression Object compile Method
	JScript Regular Expression Object exec Method
	JScript Regular Expression Object test Method
Script Engine Identification	JScript ScriptEngine Function
	JScript ScriptEngineBuildVersion Function
	JScript ScriptEngineBuildMajorVersion Function
	JScript ScriptEngineBuildMinorVersion Function
Strings	JScript String Object concat Method
	JScript String Object slice Method
	JScript String Object match Method
	JScript String Object replace Method
	JScript String Object search Method
	JScript String Object anchor Method
	JScript String Object big Method
	JScript String Object blink Method
	JScript String Object bold Method

JScript String Object fixed Method

JScript String Object fontcolor Method

JScript String Object fontsize Method

JScript String Object italics Method

JScript String Object small Method

JScript String Object strike Method

JScript String Object sub Method

JScript String Object sup Method

JScript Data Type Conversion

JScript provides automatic type conversion as the context may require. This means that if the context expects a value to be a string, for example, JScript tries to convert the value to a string.

The language has six types of data. All values have one of these types:

- **Undefined.** The undefined type has one value only, **undefined**.
- **Null.** The null type has one value only, **null**.
- **Boolean.** The Boolean type represents the two logical values, **true** and **false**.
- **String.** A string delineated by single or double quotation marks; can contain zero or more unicode characters. An empty string ("") has zero characters and length.
- **Number.** Can be an integer or floating point number according to the IEEE 754 specification. There also several special values:
- NaN, or not a Number
- Positive infinity
- Negative infinity
- Positive zero
- Negative zero
- **Object.** An object definition including its set of properties and methods.

The following table defines what happens when the context requires that JScript convert one data type into another:

Output	Input					
	Undefined	Null	Boolean	Number	String	Object
Boolean	false	false	no conversion	false if +0, -0 or NaN, otherwise	false if empty string (""), otherwise	true

				true	true	
Number	NaN	NaN	1 if true +0 if false	no conversion	If it cannot be interpreted as a number, it is interpreted as NaN	Number object
String	undefined	"null"	"true" or "false"	The absolute value of the number plus its sign, with the following exceptions: NaN returns "NaN" +0 or -0 returns "0" + infinity returns "Infinity" – infinity returns "-Infinity"	no conversion	String object
Object	runtime error	runtime error	New Boolean object	New Number object	New String object	no conversion

JScript Conditional Compilation Variables

The following predefined variables are available for conditional compilation. If a variable is not **True**, it is not defined and behaves as **NaN** when accessed.

Variable	Description
@_win32	True if running on a Win32 system.
@_win16	True if running on a Win16 system.
@_mac	True if running on an Apple Macintosh system.
@_alpha	True if running on a DEC Alpha processor.
@_x86	True if running on an Intel processor.
@_mc680x0	True if running on a Motorola 680x0 processor.
@_PowerPC	True if running on a Motorola PowerPC processor.

@_jscript	Always true.
@_jscript_build	Contains the build number of the JScript scripting engine.
@_jscript_version	Contains the JScript version number in major.minor format.

JScript Operators

Operator	Description
JScript Addition Operator (+)	Sum two numbers or perform string concatenation.
JScript Assignment Operator (=) [JScript Assignment Operator ('equals sign')	Assign a value to a variable.
JScript Bitwise AND Operator (&)	Perform bitwise AND on two expressions.
JScript Bitwise Left Shift Operator (<<)	Shift the bits of an expression to the left.
JScript Bitwise NOT Operator (~)	Perform a bitwise NOT (negation) of an expression.
JScript Bitwise OR Operator (|)	Perform bitwise OR on two expressions.
JScript Bitwise Right Shift Operator (>>)	Shift the bits of an expression right, maintaining sign.
JScript Bitwise XOR Operator (^)	Perform bitwise exclusive OR on two expressions.
JScript Comma Operator (,)	Causes expressions to be executed sequentially.
JScript Comparison Operators	Returns a Boolean value indicating the result of the comparison.
JScript Compound Assignment Operators	Combine computational operators with assignment operators to simplify expressions.
JScript Conditional (ternary) Operator (?:)	Executes one of two statements depending on a condition.
JScript Delete Operator	Deletes a property from an object, or removes an element from an array.
JScript Decrement (--) and Increment (++) Operators	Decrements a variable by one.
JScript Division Operator (/)	Divide two numbers and return a numeric result.

JScript Comparison Operators (==)	Compares two expressions for equality
JScript Comparison Operators (>)	Compares the magnitude of two expressions.
JScript Comparison Operators (>=)	Compares the magnitude or equality of two expressions.
JScript Comparison Operators (===)	Compares two expressions for equality and type.
JScript Decrement (--) and Increment (++) Operators	Increments a variable by one.
JScript Logical AND Operator (&&)	Perform a logical conjunction on two expressions.
JScript Logical NOT Operator (!)	Perform logical negation on an expression.
JScript Logical OR Operator (||)	Perform a logical disjunction on two expressions.
JScript Modulus Operator (%)	Divide two numbers and return only the remainder.
JScript Multiplication Operator (*)	Multiply two numbers.
JScript new Operator	Create a new object.
JScript Subtraction and Unary Negation Operator (-)	Indicate the negative value of a numeric expression.
Comparison (!===)	Compares two expressions for equality and type.
Subtraction (-)	Find the difference between two expressions
JScript typeof Operator	Determine the type of an expression.
JScript Unsigned Right Shift Operator (>>>)	Shift bits of an expression to the right.
JScript void Operator	Discards its operator and returns undefined.

JScript Operator Behavior

The following table describes the behavior of most JScript operators. The columns and rows represent the different types of expressions possible on either side of an operator in JScript, and the entries in the table describe the behavior.

An **E** indicates a run-time error. An **N** indicates a numeric result, or a Boolean result in the case of logical operators.

	obj	As	Ns	num	bool	Undef	null
Obj	N	E	N	N	N	E	E
As	E	E	E	E	E	E	E
Ns	N	E	N	N	N	E	E
Num	N	E	N	N	N	E	E
Bool	N	E	N	N	N	E	E
Undef	E	E	E	E	E	E	E
Null	E	E	E	E	E	E	E

obj = object, as = alphanumeric string, ns = numeric string, num = number, bool = Boolean, undef = undefined, null = null value.

JScript Operator Precedence

Operators in JScript are evaluated in a particular order. This order is known as the operator precedence. The following table lists the operators in highest-to-lowest precedence order. Operators with the same precedence are evaluated in left to right order in the expression.

Operator	Description
. [] ()	Field access, array indexing, and function calls
++ -- - ~ ! delete new typeof void	Unary operators, return data type, object creation, undefined values
* / %	Multiplication, division, modulo division
+ - +	Addition, subtraction, string concatenation
<< >> >>>	Bit shifting
< <= > >=	Less than, less than or equal, greater than, greater than or equal
== != === !==	Equality, inequality, identity, nonidentity
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
&&	Logical AND
| |	Logical OR
?:	Conditional
= *OP*=	Assignment, assignment with operation
,	Multiple evaluation

Parentheses are used to alter the order of evaluation. The expression within parentheses is fully evaluated before its value is used in the remainder of the statement.

An operator with higher precedence is evaluated before one with lower precedence. For example:

```
z = 78 * (96 + 3 + 45)
```

There are five operators in this expression: `=`, `*`, `()`, `+`, and `+`. According to precedence, they are evaluated in the following order: `()`, `*`, `+`, `+`, `=`.

Evaluation of the expression within the parentheses is first: There are two addition operators, and they have the same precedence: 96 and 3 are added together and 45 is added to that total, resulting in a value of 144.

Multiplication is next: 78 and 144 are multiplied, resulting in a value of 10998.

Assignment is last: 11232 is assigned into z.

JScript Addition Operator (+)

Used to sum two numbers or perform string concatenation.

Syntax: JScript Addition Operator (+)

```
result = expression1 + expression2
```

Arguments: JScript Addition Operator (+)

result

Any variable.

expression1

Any expression.

expression2

Any expression.

Remarks: JScript Addition Operator (+)

The underlying subtype of the expressions determines the behavior of the + operator.

If	Then
Both expressions are numeric or Boolean	Add
Both expressions are strings	Concatenate
One expression is numeric and the other is a string	Concatenate

JScript += Operator

Used to increment a variable by a specified amount

Syntax: JScript += Operator

```
result += expression
```

Arguments: JScript += Operator

result

Any variable.

expression

Any expression.

Remarks: JScript += Operator

Using this operator is exactly the same as specifying:

result = *result* + *expression*

The underlying subtype of the expression determines the behavior of the += operator:

If	Then
Both expressions are numeric or Boolean	Add
Both expressions are strings	Concatenate
One expression is numeric and the other is a string	Concatenate

For information on when a run-time error is generated by the += operator, see the Operator Behavior table.

JScript Assignment Operator (=)

Assigns a value to a variable.

Syntax: JScript Assignment Operator (=)

```
result = expression
```

Arguments: JScript Assignment Operator (=)

result

Any variable.

expression

Any numeric expression.

Remarks: JScript Assignment Operator (=)

As the = operator behaves like other operators, expressions using it have a value in addition to assigning that value into *variable*. This means that you can chain assignment operators as follows:

```
j = k = l = 0;
```

`j`, `k`, and `l` equal zero after the example statement is executed.

JScript Bitwise AND Operator (&)

Used to perform a bitwise AND on two expressions.

Syntax: JScript Bitwise AND Operator (&)

```
result = expression1 & expression2
```

Arguments: JScript Bitwise AND Operator (&)

result

Any variable.

expression1

Any expression.

expression2

Any expression.

Remarks: JScript Bitwise AND Operator (&)

The **&** operator looks at the binary representation of the values of two expressions and does a bitwise `AND` operation on them. The result of this operation behaves as follows:

```
0101 (expression1)
1100 (expression2)
----
0100 (result)
```

Any time both of the expressions have a `1` in a digit, the result has a `1` in that digit. Otherwise, the result has a `0` in that digit.

JScript &= Operator

Used to perform a bitwise AND on an expression

Syntax: JScript &= Operator

```
result &= expression
```

Arguments: JScript &= Operator

result

Any variable.

expression

Any expression.

Remarks: JScript &= Operator

Using this operator is exactly the same as specifying:

result = *result* **&** *expression*

The **&=** operator looks at the binary representation of the values of *result* and *expression* and does a bitwise AND operation on them. The output of this operation behaves like this:

```
0101   (result)
1100   (expression)
----
0100   (output)
```

Any time both *result* and *expression* have a `1` in a digit, *result* will have a `1` in that digit, otherwise *result* will have a `0` in that digit.

For information on when a run-time error is generated by the **&=** operator, see the Operator Behavior table.

JScript Bitwise Left Shift Operator (<<)

Used to shift the bits of an expression to the left.

Syntax: JScript Bitwise Left Shift Operator (<<)

```
result = expression1 << expression2
```

Arguments: JScript Bitwise Left Shift Operator (<<)

result

Any variable.

expression1

Any expression.

expression2

Any expression.

Remarks: JScript Bitwise Left Shift Operator (<<)

The **<<** operator shifts the bits of *expression1* left by the number of bits specified in *expression2*. For example:

```
var temp
temp = 14 << 2
```

The variable *temp* has a value of `56` as `14` (`00001110` in binary) shifted left two bits equals `56` (`00111000` in binary).

For information on when a run-time error is generated by the **<<** operator, see the Operator Behavior table.

JScript <<= Operator

Left shifts the value of a variable by the number of bits specified in the value of an expression and assigns the result to a variable.

Syntax: JScript <<= Operator

```
result <<= expression
```

Arguments: JScript <<= Operator

result

Any variable.

expression

Any expression.

Remarks: JScript <<= Operator

Using this operator is exactly the same as specifying:

```
result = result << expression
```

The <<= operator shifts the bits of *result* left by the number of bits specified in *expression.* For example:

```
var temp

temp = 14

temp <<= 2
```

The variable *temp* has a value of `56` as `14` (`00001110` in binary) shifted left two bits equals `56` (`00111000`) in binary. Bits are filled in with zeroes when shifting.

For information on when a run-time error is generated by the **<<=** operator, see the Operator Behavior table.

JScript Bitwise NOT Operator (~)

Used to perform a bitwise NOT (negation) on an expression.

Syntax: JScript Bitwise NOT Operator (~)

```
result = ~ expression
```

Arguments: JScript Bitwise NOT Operator (~)

result

Any variable.

expression

Any expression.

Remarks: JScript Bitwise NOT Operator (~)

All unary operators, such as the **~** operator, evaluate expressions as follows:

- If applied to undefined or null expressions, a run-time error is raised.
- Objects are converted to strings.
- Strings are converted to numbers if possible. If not, a run-time error is raised.
- Boolean values are treated as numbers (`0` if false, `1` if true).

The operator is applied to the resulting number.

The **~** operator looks at the binary representation of the values of the expression and does a bitwise negation operation on it. The result of this operation behaves as follows:

```
0101 (expression)
----
1010 (result)
1011
```

Any digit that is a `1` in the expression becomes a `0` in the result. Any digit that is a `0` in the expression becomes a `1` in the result.

JScript Bitwise OR Operator (|)

Used to perform a bitwise OR on two expressions.

Syntax: JScript Bitwise OR Operator (|)

```
result = expression1 | expression2
```

Arguments: JScript Bitwise OR Operator (|)

result

Any variable.

expression1

Any expression.

expression2

Any expression.

Remarks: JScript Bitwise OR Operator (|)

The **|** operator looks at the binary representation of the values of two expressions and does a bitwise `OR` operation on them. The result of this operation behaves as follows:

```
0101 (expression1)
```

```
1100 (expression2)
----
1101 (result)
```

Any time either of the expressions has a `1` in a digit, the result will have a `1` in that digit. Otherwise, the result will have a `0` in that digit.

For information on when a run-time error is generated by the **|** operator, see the Operator Behavior table.

JScript |= Operator

Used to perform a bitwise `OR` on an expression.

Syntax: JScript |= Operator

```
result |= expression
```

Arguments: JScript |= Operator

result

Any variable.

expression

Any expression.

Remarks: JScript |= Operator

Using this operator is exactly the same as specifying:

```
result = result | expression
```

The **|=** operator looks at the binary representation of the values of *result* and *expression* and does a bitwise `OR` operation on them. The output of this operation behaves like this:

```
0101  (result)
1100  (expression)
----
1101  (output)
```

Any time either *result* or *expression* has a `1` in a digit, *result* will have a 1 in that digit, otherwise *result* will have a `0` in that digit.

For information on when a run-time error is generated by the `|=` operator, see the Operator Behavior table.

JScript Bitwise Right Shift Operator (>>)

Used to shift the bits of an expression to the right, maintaining sign.

Syntax: JScript Bitwise Right Shift Operator (>>)

```
result = expression1 >> expression2
```

Arguments: JScript Bitwise Right Shift Operator (>>)

result

Any variable.

expression1

Any expression.

expression2

Any expression.

Remarks: JScript Bitwise Right Shift Operator (>>)

The **>>** operator shifts the bits of *expression1* right by the number of bits specified in *expression2*. The sign bit of *expression1* is used to fill the digits from the left. Digits shifted off the right are discarded. For example, after the following code is evaluated, *temp* has a value of `-4`: `14` (`11110010` in binary) shifted right two bits equals `-4` (`11111100` in binary).

```
var temp

temp = -14 >> 2
```

For information on when a run-time error is generated by the **>>** operator, see the Operator Behavior table.

JScript >>= Operator

Used to shift the bits of an expression to the right, preserving sign.

Syntax: JScript >>= Operator

```
result >>= expression
```

Arguments: JScript >>= Operator

result

Any variable.

expression

Any expression.

Remarks: JScript >>= Operator

Using this operator is exactly the same as specifying:

```
result = result >> expression
```

The **>>=** operator shifts the bits of *result* right by the number of bits specified in *expression*. The sign bit of *result* is used to fill the digits from the left. Digits shifted off the right are discarded.

For example, after the following code is evaluated, *temp* has a value of `-4: 14` (`11110010` in binary) shifted right two bits equals `-4` (`11111100` in binary).

```
var temp

temp = -14

temp <<= 2
```

For information on when a run-time error is generated by the **>>=** operator, see the Operator Behavior table.

JScript Bitwise XOR Operator (^)

Used to perform a bitwise exclusive `OR` on two expressions.

Syntax: JScript Bitwise XOR Operator (^)

```
result = expression1 ^ expression2
```

Arguments: JScript Bitwise XOR Operator (^)

result

Any variable.

expression1

Any expression.

expression2

Any expression.

Remarks: JScript Bitwise XOR Operator (^)

The **^** operator looks at the binary representation of the values of two expressions and does a bitwise exclusive `OR` operation on them. The result of this operation behaves as follows:

```
0101 (expression1)

1100 (expression2)

----

1001 (result)
```

When one, and only one, of the expressions has a `1` in a digit, the result has a `1` in that digit. Otherwise, the result has a `0` in that digit.

For information on when a run-time error is generated by the **^** operator, see the Operator Behavior table.

JScript ^= Operator

Used to perform a bitwise exclusive `OR` on an expression.

Syntax: JScript ^= Operator

```
result ^= expression
```

Arguments: JScript ^= Operator

result

Any variable.

expression

Any expression.

Remarks: JScript ^= Operator

Using this operator is exactly the same as specifying:

```
result = result ^ expression
```

The **^=** operator looks at the binary representation of the values of *result* and *expression* and does a bitwise `OR` operation on them. The output of this operation behaves like this:

```
0101   (result)
1100   (expression)
----
1001   (output)
```

When one, and only one, of *result* or *expression* have a `1` in a digit, *result* will have a 1 in that digit, otherwise *result* will have a `0` in that digit.

For information on when a run-time error is generated by the **^=** operator, see the Operator Behavior table.

JScript Comma Operator (,)

Causes two expressions to be executed sequentially.

Syntax: JScript Comma Operator (,)

```
expression1, expression2
```

Arguments: JScript Comma Operator (,)

expression1

Any expression.

expression2

Any expression.

Remarks: JScript Comma Operator (,)

The , operator causes the expressions on either side of it to be executed in left-to-right order, and obtains the value of the expression on the right. The most common use for the , operator is in the increment expression of a **for** loop. For example:

```
for (i = 0; i < 10; i++, j++)
{
k = i + j;
}
```

The **for** statement only allows a single expression to be executed at the end of every pass through a loop. The , operator is used to allow multiple expressions to be treated as a single expression, thereby getting around the restriction.

JScript Comparison Operators

Returns a Boolean value indicating the result of the comparison.

Syntax: JScript Comparison Operators

```
expression1 comparisonoperator expression2
```

Arguments: JScript Comparison Operators

expression1

Any expression.

comparisonoperator

Any comparison operator.

expression2

Any expression.

Remarks: JScript Comparison Operators

When comparing strings, JScript uses the Unicode character value of the **String** expression. The following is how the different groups of operators behave depending on the types and values of expression1 and expression2:

Relational (<, >, <=, =)

Attempt to convert both *expression1* and *expression2* into numbers.

- If both expressions are strings, do a lexicographical string comparison.
- If either expression is **NaN**, return **false**.
- Negative zero equals Positive zero.
- Negative Infinity is less than everything including itself.
- Positive Infinity is greater than everything including itself.

Equality (==, !=)

If the types of the two expressions are different, attempt to convert them to string, number, or Boolean.

- **NaN** is not equal to anything including itself.
- Negative zero equals positive zero.
- **Null** equals both **null** and undefined.
- Values are considered equal if they are identical strings, numerically equivalent numbers, the same object, identical Boolean values, or (if different types) they can be coerced into one of these situations.
- Every other comparison is considered unequal.

Identity (===. !==)

These operators behave identically to the equality operators except no type conversion is done, and the types must be the same to be considered equal.

JScript Compound Assignment Operators

The compound assignment operators combine computational operators with assignment operators to simplify expressions. The following compound assignment operators are available:

Operator	Description
JScript += Operator (JScript 'plus equals' Operator)	Compound addition operator.
JScript &= Operator (JScript &'equals' Operator)	Compound bitwise AND operator.
JScript |= Operator (JScript |'equals' Operator)	Compound bitwise OR operator.
JScript ^= Operator (JScript ^'equals' Operator)	Compound exclusive OR operator.
JScript /= Operator (JScript /'equals' Operator)	Compound division operator.
JScript <<= Operator (JScript <<'equals'Operator)	Compound left shift operator.
JScript %= Operator (JScript %'equals' Operator)	Compound modulus operator.
JScript *= Operator (JScript *'equal' Operator)	Compound multiplication operator.
JScript >>= Operator (JScript >>'equals' Operator)	Compound right shift operator.
JScript -= Operator (JScript -'equal' Operator)	Compound subtraction operator.
JScript >>>= Operator (JScript >>>'equals' Operator)	Compound unsigned right-shift operator.

JScript Conditional (ternary) Operator (?:)

Executes one of two statements depending on a condition.

Syntax: JScript Conditional (ternary) Operator (?:)

```
test ? statement1 : statement2
```

Arguments: JScript Conditional (ternary) Operator (?:)

test

Any Boolean expression.

statement1

A statement executed if test is **true**. May be a compound statement.

statement2

A statement executed if test is **false**. May be a compound statement.

Remarks: JScript Conditional (ternary) Operator (?:)

The **?:** operator is a shortcut for an **if...else** statement. It is typically used as part of a larger expression where an **if...else** statement would be awkward. For example:

```
var now = new Date();
var greeting = "Good" + ((now.getHours() 17) ? " evening." :
" day.");
```

The example creates a string containing "Good evening." if it is after 6 pm. The equivalent code using an **if...else** statement would look as follows:

```
var now = new Date();
var greeting = "Good";
if (now.getHours() 17)
greeting += " evening.";
else
greeting += " day.";
```

JScript delete Operator

Deletes a property from an object, or removes an element from an array.

Syntax: JScript delete Operator

```
delete expression
```

Arguments: JScript delete Operator

expression

A valid JScript expression that usually (but does not have to) results in a property name or array element.

Remarks: JScript delete Operator

If the result of *expression* is an object, and the property specified in *expression* exists, and the object will not allow it to be deleted, **false** is returned.

In all other cases **true** is returned.

JScript Decrement (--) and Increment (++) Operators

++ and **--** operators are used to increment or decrement a variable by one.

Syntax: JScript Decrement (--) and Increment (++) Operators

```
result = ++variable

result = --variable

result = variable++

result = variable--
```

Syntax 2: JScript Decrement (--) and Increment (++) Operators

```
++variable

--variable

variable++

variable--
```

Arguments: JScript Decrement (--) and Increment (++) Operators

result

Any variable.

variable

Any variable.

Remarks: JScript Decrement (--) and Increment (++) Operators

The increment and decrement operators are used as a shortcut to modify the value stored in a variable. The value of an expression containing one of these operators depends on whether the operator comes before or after the variable:

```
var j, k;

k = 2;

j = ++k;
```

j is assigned the value 3, as the increment occurs before the expression is evaluated. Contrast the following example:

```
var j, k;
k = 2;
j = k++;
```

Here, *j* is assigned the value 2, as the increment occurs after the expression is evaluated.

JScript Division Operator (/)

Used to divide two numbers and return a numeric result.

Syntax: JScript Division Operator (/)

```
result = number1 / number2
```

Arguments: JScript Division Operator (/)

result

Any numeric variable.

number1

Any numeric expression.

number2

Any numeric expression.

Remarks: JScript Division Operator (/)

For information on when a run-time error is generated by the **/** operator, see the Operator Behavior table.

JScript /= Operator

Used to divide a variable by an expression.

Syntax: JScript /= Operator

```
result /= expression
```

Arguments: JScript /= Operator

result

Any variable.

expression

Any expression.

Remarks: JScript /= Operator

Using this operator is exactly the same as specifying:

```
result = result / expression
```

For information on when a run-time error is generated by the **/=** operator, see the Operator Behavior table.

JScript Logical AND Operator (&&)

Used to perform a logical conjunction on two expressions.

Syntax: JScript Logical AND Operator (&&)

```
result = expression1 && expression2
```

Arguments: JScript Logical AND Operator (&&)

result

Any variable.

expression1

Any expression.

expression2

Any expression.

Remarks: JScript Logical AND Operator (&&)

If, and only if, both expressions evaluate to **True**, *result* is **True**. If either expression evaluates to **False**, *result* is **False**.

For information on when a run-time error is generated by the **&&** operator, see the Operator Behavior table.

JScript uses the following rules for converting non-Boolean values to Boolean values:

- All objects are considered true.
- Strings are considered false if and only if they are empty.
- **Null** and undefined are considered false.
- Numbers are false if and only if they are zero.

JScript Logical NOT Operator (!)

Used to perform logical negation on an expression.

Syntax: JScript Logical NOT Operator (!)

```
result = !expression
```

Arguments: JScript Logical NOT Operator (!)

result

Any variable.

expression

Any expression.

Remarks: JScript Logical NOT Operator (!)

The following table illustrates how result is determined.

If *expression* is	Then *result* is
True	False
False	True

All unary operators, such as the **!** operator, evaluate expressions as follows:

- If applied to undefined or **null** expressions, a run-time error is raised.
- Objects are converted to strings.
- Strings are converted to numbers if possible. If not, a run-time error is raised.
- Boolean values are treated as numbers (`0` if false, `1` if true).

The operator is applied to the resulting number.

For the **!** operator, if *expression* is nonzero, *result* is zero. If *expression* is zero, *result* is `1`.

JScript Logical OR Operator (||)

Used to perform a logical disjunction on two expressions.

Syntax: JScript Logical OR Operator (||)

```
result = expression1 || expression2
```

Arguments: JScript Logical OR Operator (||)

result

Any variable.

expression1

Any expression.

expression2

Any expression.

Remarks: JScript Logical OR Operator (||)

If either or both expressions evaluate to **True**, *result* is **True**. The following table illustrates how *result* is determined:

If *expression1* is	And *expression2* is	The *result* is
True	True	True
True	False	True
False	True	True
False	False	False

For information on when a run-time error is generated by the || operator, see the Operator Behavior table.

JScript uses the following rules for converting non-Boolean values to Boolean values:

- All objects are considered true.
- Strings are considered false if and only if they are empty.
- **Null** and undefined are considered false.
- Numbers are false if and only if they are 0.

JScript Modulus Operator (%)

Used to divide two numbers and return only the remainder.

Syntax: JScript Modulus Operator (%)

```
result = number1 % number2
```

Arguments: JScript Modulus Operator (%)

result

Any variable.

number1

Any numeric expression.

number2

Any numeric expression.

Remarks: JScript Modulus Operator (%)

The modulus, or remainder, operator divides *number1* by *number2* (rounding floating-point numbers to integers) and returns only the remainder as *result.* For example, in the following expression, *A* (which is *result*) equals 5.

```
A = 19 % 6.7
```

For information on when a run-time error is generated by the **%** operator, see the Operator Behavior table.

JScript %= Operator

Used to divide two numbers and return only the remainder

Syntax: JScript %= Operator

```
result %= expression
```

Arguments: JScript %= Operator

result

Any variable.

expression

Any expression.

Remarks: JScript %= Operator

Using this operator is exactly the same as specifying:

```
result = result % expression
```

For information on when a run-time error is generated by the **%=** operator, see the Operator Behavior table.

JScript Multiplication Operator (*)

Used to multiply two numbers.

Syntax: JScript Multiplication Operator (*)

```
result = number1 * number2
```

Arguments: JScript Multiplication Operator (*)

result

Any variable.

number1

Any expression.

number2

Any expression.

Remarks: JScript Multiplication Operator (*)

For information on when a run-time error is generated by the * operator, see the Operator Behavior table.

JScript *= Operator

Used to multiply one number by another number.

*Syntax: JScript *= Operator*

```
result *= expression
```

*Arguments: JScript *= Operator*

result

Any variable.

expression

Any expression.

*Remarks: JScript *= Operator*

Using this operator is exactly the same as specifying:

```
result = result * expression
```

For information on when a run-time error is generated by the *= operator, see the Operator Behavior table.

JScript new Operator

Creates a new object.

Syntax: JScript new Operator

```
new constructor[(arguments)]
```

Arguments: JScript new Operator

constructor

The constructor argument calls object's constructor. The parentheses can be omitted if the constructor takes no arguments.

arguments

Optional. Any arguments to be passed to the new object's constructor.

Remarks: JScript new Operator

The **new** operator performs the following tasks:

- It creates an object with no members.
- It calls the constructor for that object, passing a pointer to the newly created object as the **this** pointer.
- The constructor then initializes the object according to the arguments passed to the constructor.

These are examples of valid uses of the **new** operator:

```
my_object = new Object;
```

```
my_array = new Array();
```

```
my_date = new Date("Jan 5 1996");
```

JScript Subtraction and Unary Negation Operator (-)

Used to find the difference between two numbers or to indicate the negative value of a numeric expression.

Syntax 1: JScript Subtraction and Unary Negation Operator (-)

```
result = number1 - number2
```

Syntax 2: JScript Subtraction and Unary Negation Operator (-)

```
-number
```

Arguments: JScript Subtraction and Unary Negation Operator (-)

result

Any numeric variable.

number

Any numeric expression.

number1

Any numeric expression.

number2

Any numeric expression.

Remarks: JScript Subtraction and Unary Negation Operator (-)

In Syntax 1, the - operator is the arithmetic subtraction operator used to find the difference between two numbers. In Syntax 2, the - operator is used as the unary negation operator to indicate the negative value of an expression.

For information on when a run-time error is generated by Syntax 1, see the Operator Behavior table.

For Syntax 2, as for all unary operators, expressions are evaluated as follows:

- If applied to undefined or **null** expressions, a run-time error is raised.
- Objects are converted to strings.
- Strings are converted to numbers if possible. If not, a run-time error is raised.
- Boolean values are treated as numbers (0 if false, 1 if true).

The operator is applied to the resulting number. In Syntax 2, if the resulting number is nonzero, *result* is equal to the resulting number with its sign reversed. If the resulting number is zero, *result* is zero.

JScript -= Operator

Used to subtract the value of an expression from a variable.

Syntax: JScript -= Operator

```
result -= expression
```

Arguments: JScript -= Operator

result

Any variable.

expression

Any expression.

Remarks: JScript -= Operator

Using this operator is exactly the same as specifying:

```
result = result - expression
```

For information on when a run-time error is generated by the **-=** operator, see the Operator Behavior table.

JScript typeof Operator

Used to determine the type of an expression.

Syntax: JScript typeof Operator

```
typeof [ ( ] expression [ ) ] ;
```

Arguments: JScript typeof Operator

expression

Any expression for which type information is sought.

Remarks: JScript typeof Operator

The **typeof** operator returns type information as a string. There are six possible values that **typeof** returns: "number," "string," "boolean," "object," "function," and "undefined."

The parentheses are optional in the **typeof** syntax.

JScript Unsigned Right Shift Operator (>>>)

Used to make an unsigned right shift of the bits in an expression.

Syntax: JScript Unsigned Right Shift Operator (>>>)

```
result = expression1 >>> expression2
```

Arguments: JScript Unsigned Right Shift Operator (>>>)

result

Any variable.

expression1

Any expression.

expression2

Any expression.

Remarks: JScript Unsigned Right Shift Operator (>>>)

The **>>>** operator shifts the bits of *expression1* right by the number of bits specified in *expression2*. Zeroes are filled in from the left. Digits shifted off the right are discarded. For example:

```
var temp

temp = -14 >> 2
```

The variable temp has a value of 1073741820 as -14 (11111111 11111111 11111111 11110010 in binary) shifted right two bits equals 1073741820 (00111111 11111111 11111111 11111100 in binary).

For information on when a run-time error is generated by the **>>>** operator, see the Operator Behavior table.

JScript >>>= Operator

Used to make an unsigned right shift of the bits in a variable.

Syntax: JScript >>>= Operator

```
result >>>= expression
```

Arguments: JScript >>>= Operator

result

Any variable.

expression

Any expression

Remarks: JScript >>>= Operator

Using this operator is exactly the same as specifying:

```
result = result >>> expression
```

The **>>>=** operator shifts the bits of *result* right by the number of bits specified in *expression*. Zeroes are filled in from the left. Digits shifted off the right are discarded. For example:

```
var temp

temp = -14

temp >>>= 2
```

The variable *temp* has a value of 1073741820 as –14 (11111111 11111111 11111111 11110010 in binary) shifted right two bits equals 1073741820 (00111111 11111111 11111111 11111100 in binary).

For information on when a run-time error is generated by the **>>>=** operator, see the Operator Behavior table.

JScript void Operator

Discards its operator and returns undefined.

Syntax: JScript void Operator

```
void expression
```

Arguments: JScript void Operator

expression

Any valid JScript expression.

Remarks: JScript void Operator

The **void** operator evaluates its expression, and returns undefined. It is most useful in situations where you want an expression evaluated but do not want the results visible to the remainder of the script.

JScript Functions

Function	Description
JScript Getobject Function	Returns a reference to an Automation object from a file. *This is a client-side only function.*
JScript ScriptEngine Function	Returns a string representing the scripting language in use.
JScript ScriptEngineBuildVersion Function	Returns the build version number of the script engine in use.
JScript ScriptEngineBuildMajorVersion Function	Returns the major version number of the script engine in use.
JScript ScriptEngineBuildMinorVersion Function	Returns the minor version number of the script engine in use.

JScript Getobject Function

Returns a reference to an Automation object from a file. *This is a client-side only function.*

Syntax: JScript Getobject Function

```
GetObject([pathname] [, class])
```

Arguments: JScript Getobject Function

pathname

An optional full path and name of the file containing the object to retrieve. If *pathname* is omitted, *class* is required.

class

An optional class of the object. The *class* argument uses the syntax *appname.objectype* and has these parts:

appname

A name of the application providing the object. Required.

objectype

A type or class of the object to create. Required.

Remarks: JScript Getobject Function

Use the **GetObject** function to access an Automation object from a file and assign the object to an object variable. Assign the object returned by **GetObject** to the object variable. For example:

```
var CADObject;

CADObject = GetObject("C:\\CAD\\SCHEMA.CAD");
```

When this code is executed, the application associated with the specified pathname is started, and the object in the specified file is activated. If *pathname* is a zero-length string (""), **GetObject** returns a new object instance of the specified type. If the *pathname* argument is omitted, **GetObject** returns a currently active object of the specified type. If no object of the specified type exists, an error occurs.

Some applications allow you to activate part of a file. Add an exclamation point (!) to the end of the file name and follow it with a string that identifies the part of the file you want to activate. For information on how to create this string, see the documentation for the application that created the object.

For example, in a drawing application you might have multiple layers to a drawing stored in a file. You could use the following code to activate a layer within a drawing called SCHEMA.CAD:

```
var LayerObject = GetObject("C:\\CAD\\SCHEMA.CAD!Layer3");
```

If you don't specify the object's class, Automation determines the application to start and the object to activate, based on the file name you provide. Some files, however, may support more than one class of object. For example, a drawing might support three different types of objects: an

Application object, a **Drawing** object, and a **Toolbar** object, all of which are part of the same file. To specify which object in a file you want to activate, use the optional *class* argument. For example:

```
var MyObject;
MyObject = GetObject("C:\\DRAWINGS\\SAMPLE.DRW",
  "FIGMENT.DRAWING");
```

In the preceding example, `FIGMENT` is the name of a drawing application and `DRAWING` is one of the object types it supports. Once an object is activated, you reference it in code using the object variable you defined. In the preceding example, you access properties and methods of the new object using the object variable `MyObject`. For example:

```
MyObject.Line(9, 90);
MyObject.InsertText(9, 100, "Hello, world.");
MyObject.SaveAs("C:\\DRAWINGS\\SAMPLE.DRW");
```

Note

Use the **GetObject** function when there is a current instance of the object or if you want to create the object with a file already loaded. If there is no current instance, and you don't want the object started with a file loaded, use the **ActiveXObject** object.

If an object has registered itself as a single-instance object, only one instance of the object is created, no matter how many times **ActiveXObject** is executed. With a single-instance object, **GetObject** always returns the same instance when called with the zero-length string ("") syntax, and it causes an error if the *pathname* argument is omitted.

JScript ScriptEngine Function

Returns a string representing the scripting language in use.

Syntax: JScript ScriptEngine Function

```
ScriptEngine( );
```

Return Values: JScript ScriptEngine Function

The **ScriptEngine** function can return any of the following strings:

JScript

Microsoft JScript is the current script engine.

VBA

Microsoft Visual Basic for Applications is the current script engine.

VBScript

Microsoft Visual Basic Scripting Edition is the current script engine.

JScript ScriptEngineBuildVersion Function

Returns the build version number of the script engine in use.

Syntax: JScript ScriptEngineBuildVersion Function

```
ScriptEngineBuildVersion( );
```

Return Values: JScript ScriptEngineBuildVersion Function

The return value corresponds directly to the version information contained in the DLL for the scripting language in use.

JScript ScriptEngineBuildMajorVersion Function

Returns the major version number of the script engine in use.

Syntax: JScript ScriptEngineBuildMajorVersion Function

```
ScriptEngineMajorVersion( );
```

Return Values: JScript ScriptEngineBuildMajorVersion Function

The return value corresponds directly to the version information contained in the DLL for the scripting language in use.

JScript ScriptEngineBuildMinorVersion Function

Returns the minor version number of the script engine in use.

Syntax: JScript ScriptEngineBuildMinorVersion Function

```
ScriptEngineMinorVersion( );
```

Return Values: JScript ScriptEngineBuildMinorVersion Function

The return value corresponds directly to the version information contained in the DLL for the scripting language in use.

JScript Statements

Statement	Description
JScript break Statement	Terminates the current loop or associated statement.
JScript @cc_on Statement	Activates conditional compilation support.
JScript Comment Statements	Causes comments to be ignored by the JScript parser.
JScript continue Statement	Stops the current iteration of a loop and starts a new iteration.

JScript do. . . while Statement	Executes a block once, and then repeats execution of the loop until a conditional expression evaluates to **False**.
JScript for Statement	Executes a block of statements as long as a specified condition is **True**.
JScript for. . . in Statement	Executes a statement for each element of an object or array.
JScript function Statement	Declares a new function.
JScript @if Statement	Conditionally executes a group of statements depending on the value of an expression.
JScript if. . . else Statement	Conditionally executes a group of statements depending on the value of an expression.
JScript Labeled Statement	Provides an identifier for a statement.
JScript return Statement	Exits from the current function and returns a value from that function.
JScript @set Statement	Allows creation of variables used in conditional compilation statements.
JScript switch Statement	Enables the execution of one or more statements when a specified expression's value matches a label.
JScript this Statement	Refers to the current object.
JScript var Statement	Declares a variable.
JScript with Statement	Establishes the default object for a statement.
JScript while Statement	Executes a statement until a specified condition is **False**.

JScript break Statement

Terminates the current loop, or if in conjunction with a *label*, terminates the associated statement.

Syntax: JScript break Statement

```
break [label];
```

Arguments: JScript break Statement

label

Optional argument that specifies the label of the statement you are breaking from.

Remarks: JScript break Statement

You typically use the **break** statement in **switch** statements and **while**, **for**, **for…in**, or **do…while** loops. You most commonly use the *label* argument in **switch** statements, but it can be used in any statement, whether simple or compound.

Executing the **break** statement exits from the current loop or statement, and begins script execution with the statement immediately following.

JScript @cc_on Statement

Activates conditional compilation support.

Syntax: JScript @cc_on Statement

```
@cc_on
```

Remarks: JScript @cc_on Statement

The **@cc_on** statement activates conditional compilation in the scripting engine.

It is strongly recommended that you use the **@cc_on** statement in a comment, so that browsers that do not support conditional compilation will accept your script as valid syntax:

```
/*@cc_on*/
...
(remainder of script)
```

Alternatively, an **@if** or **@set** statement outside of a comment also activates conditional compilation.

JScript Comment Statement

Creates comments that are ignored by the JScript parser.

Syntax 1: JScript Comment Statement

Single-line Comment:

```
// comment
```

Syntax 2: JScript Comment Statement

Multiline Comment:

```
/*
comment
*/
```

Arguments: JScript Comment Statement

comment

The text of any comment you want to include in your script.

Syntax 3: JScript Comment Statement

```
//@CondStatement
```

Syntax 4: JScript Comment Statement

```
/*@
CondStatement
@*/
```

Arguments: JScript Comment Statement

CondStatement

Conditional compilation code to be used if conditional compilation is activated. If Syntax 3 is used, there can be no space between the "//" and "@" characters.

Remarks: JScript Comment Statement

Use comments to keep parts of a script from being read by the JScript parser. You can use comments to include explanatory remarks in a program.

If Syntax 1 is used, the parser ignores any text between the comment marker and the end of the line. If Syntax 2 is used, it ignores any text between the beginning and end markers.

Syntax 3 and Syntax 4 are used to support conditional compilation while retaining compatibility with browsers that do not support that feature. These browsers treat those forms of comments as Syntax 1 and Syntax 2 respectively.

JScript continue Statement

Stops the current iteration of a loop, and starts a new iteration.

Syntax: JScript continue Statement

```
continue [label];
```

Arguments: JScript continue Statement

label

Optional argument that specifies the statement to which **continue** applies.

Remarks: JScript continue Statement

You can use the **continue** statement only inside a **while**, **do…while**, **for**, or **for…in** loop. Executing the **continue** statement stops the current iteration of the loop and continues program flow with the beginning of the loop. This has the following effects on the different types of loops:

- **while** and **do...while** loops test their condition, and if true, execute the loop again.
- **for** loops execute their increment expression, and if the test expression is true, execute the loop again.
- **for...in** loops proceed to the next field of the specified variable and execute the loop again.

JScript do. . . while Statement

Executes a statement block once, and then repeats execution of the loop until a condition expression evaluates to **false**.

Syntax: JScript do. . . while Statement

```
do
statement
while (expression) ;
```

Arguments: JScript do. . . while Statement

statement

The statement to be executed if *expression* is **True**. Can be a compound statement.

expression

An expression that can be coerced to Boolean **True** or **False**. If *expression* is **True**, the loop is executed again. If *expression* is **False**, the loop is terminated.

Remarks: JScript do. . . while Statement

The value of *expression* is not checked until after the first iteration of the loop, guaranteeing that the loop is executed at least once. Thereafter, it is checked after each succeeding iteration of the loop.

JScript for Statement

Executes a block of statements for as long as a specified condition is true.

Syntax: JScript for Statement

```
for (initialization; test; increment)
statement
```

Arguments: JScript for Statement

initialization

An expression. This expression is executed only once, before the loop is executed.

test

A Boolean expression. If *test* is **True**, *statement* is executed. If *test* if **False**, the loop is terminated.

increment

An expression. The increment expression is executed at the end of every pass through the loop.

statement

The statement to be executed if *test* is **True**. Can be a compound statement.

Remarks: JScript for Statement

You usually use a **for** loop when the loop is to be executed a specific number of times as the following example demonstrates:

```
/* i is set to 0 at start, and is incremented by 1 at the end
of each iteration. Loop terminates when i is not less
than 10 before a loop iteration. */
for (i = 0; i < 10; i++)
{
j *= i;
}
```

JScript for. . . in Statement

Executes a statement for each element of an object or array.

Syntax: JScript for. . . in Statement

```
for (variable in [object | array])
statement
```

Arguments: JScript for. . . in Statement

variable

A variable that can hold any of the elements of *object*.

object, array

An object or array over which to iterate.

statement

The statement to be executed for each member of *object*. Can be a compound statement.

Remarks: JScript for. . . in Statement

Before each iteration of a loop, *variable* is assigned the next element of *object*. You can then use it in any of the statements inside the loop exactly as if you were using the element of *object*.

When iterating over an object, there is no way to determine or control the order in which the members of the object are assigned to *variable*.

JScript function Statement

Declares a new function.

Syntax: JScript function Statement

```
function functionname([argument1 [, argument2 [, ...argumentn]]])
{
statements
}
```

Arguments: JScript function Statement

functionname

The name of the function.

argument1...argumentn

An optional, comma-separated list of arguments the function understands.

statements

One or more JScript statements.

Remarks: JScript function Statement

Use the **function** statement to declare a function for later use. The code contained in *statements* is not executed until the function is called from elsewhere in the script.

JScript @if Statement

Conditionally executes a group of statements, depending on the value of an expression.

Syntax: JScript @if Statement

```
@if (condition1)
text1
[@elif (condition2)
text2]
[@else
text3]
@end
```

Arguments: JScript @if Statement

condition1, *condition2*

An expression that can be coerced into a Boolean expression.

text1

The text to be parsed if *condition1* is **True**.

text2

The text to be parsed if *condition1* is **False** and *condition2* is **True**.

text3

The text to be parsed if both *condition1* and *condition2* are **False**.

Remarks: JScript @if Statement

When writing an **@if** statement, its clauses do not have to appear on separate lines. In addition, you may use multiple **@elif** clauses. If you do, all must come before an **@else** clause.

You commonly use the **@if** statement to determine which text among several options should be used for text output. For example:

```
alert(@if (@_win32) "using Windows NT or Windows 95"
@else "using Windows 3.1" @end)
```

JScript if. . . else Statement

Conditionally executes a group of statements, depending on the value of an expression.

Syntax: JScript if. . . else Statement

```
if (condition)
statement1
[else
statement2]
```

Arguments: JScript if. . . else Statement

condition

A Boolean expression. If *condition* is **null** or undefined, *condition* is treated as **false**.

statement1

The statement to be executed if *condition* is **True**. Can be a compound statement.

statement2

The statement to be executed if *condition* is **False**. Can be a compound statement.

Remarks: JScript if. . . else Statement

It is generally good practice to enclose *statement1* and *statement2* in braces ({}) for clarity and to avoid inadvertent errors. In the following example, you may intend that the **else** be used with the first **if** statement, but it is used with the second one.

```
if (x == 5)
if (y == 6)
z = 17;
else
```

```
z = 20;
```

Changing the code in the following manner eliminates any ambiguities:

```
if (x == 5)
{
if (y == 6)
z = 17;
}
else
z = 20;
```

Similarly, if you want to add a statement to *statement1*, and you don't use braces, you can accidentally create an error:

```
if (x == 5)
z = 7;
q = 42;
else
z = 19;
```

In this case, there is a syntax error, as there is more than one statement between the **if** and **else** statements. Putting braces around the statements between the **if** and **else** is required.

JScript Labeled Statement

Provides an identifier for a statement.

Syntax: JScript Labeled Statement

```
label :
statement
```

Arguments: JScript Labeled Statement

label

A unique identifier used when referring to the labeled statement.

statement

The statement associated with *label*. May be a compound statement.

Remarks: JScript Labeled Statement

Labels are used by the **break** and **continue** statements to specify the statement to which the **break** and **continue** apply.

JScript return Statement

Exits from the current function and returns a value from that function.

Syntax: JScript return Statement

```
return [expression];
```

Arguments: JScript return Statement

expression

The value to be returned from the function. If omitted, the function does not return a value.

Remarks: JScript return Statement

You use the **return** statement to stop execution of a function and return the value of *expression*. If *expression* is omitted, or no **return** statement is executed from within the function, the expression that called the current function is assigned the value undefined.

JScript @set Statement

Allows creation of variables used in conditional compilation statements.

Syntax: JScript @set Statement

```
@set @varname = term
```

Arguments: JScript @set Statement

varname

A valid JScript variable name. Must be preceded by an "@" character at all times.

term

Zero or more unary operators followed by either a constant, conditional compilation variable, or parenthesized expression.

Remarks: JScript @set Statement

Numeric and Boolean variables are supported for conditional compilation. Strings are not. Variables created using **@set** are generally used in conditional compilation statements, but can be used anywhere in JScript code.

Examples of variable declarations look like this:

```
@set @myvar1 = 12

@set @myvar2 = (@myvar1 * 20)

@set @myvar3 = @_jscript_version
```

The following operators are supported in parenthesized expressions:

```
! ~

* / %
```

```
+ -

<< >> >>>

< <= > >=

== != === !==

& ^ |

&& ||
```

If a variable is used before it has been defined, its value is **NaN**. **NaN** can be checked for using the **@if** statement:

```
@if (@newVar != @newVar)

...
```

This works because **NaN** is the only value not equal to itself.

JScript switch Statement

Enables the execution of one or more statements when a specified expression's value matches a label.

Syntax: JScript switch Statement

```
switch (expression) {

case label :

statementlist

case label :

statementlist

...

default :

statementlist

}
```

Arguments: JScript switch Statement

expression

The expression to be evaluated.

label

An identifier to be matched against *expression*. If *label* = = = *expression*, execution starts with the *statementlist* immediately after the colon and continues until it encounters either a **break** statement, which is optional, or the end of the **switch** statement.

statementlist

One or more statements to be executed.

Remarks: JScript switch Statement

Use the **default** clause to provide a statement to be executed if none of the label values matches *expression*. It can appear anywhere within the **switch** code block.

Zero or more *label* blocks may be specified. If no *label* matches the value of *expression*, and a default case is not supplied, no statements are executed.

Execution flows through a switch statement as follows:

- Evaluate *expression* and look at *label* in order until a match is found.
- If a *label* value equals *expression*, execute its accompanying *statementlist*. Continue execution until a **break** statement is encountered, or the **switch** statement ends. This means that multiple *label* blocks are executed if a **break** statement is not used.
- If no *label* equals *expression*, go to the **default** case. If there is no **default** case, go to last step.
- Continue execution at the statement following the end of the **switch** code block.

The following example tests an object for its type:

```
function MyObject() {
...}
switch (object.constructor){
case Date:
...
case Number:
...
case String:
...
case MyObject:
...
default:
...
}
```

JScript this Statement

Refers to the current object.

Syntax: JScript this Statement

```
this.property
```

Remarks: JScript this Statement

The **this** keyword is typically used in object constructors to refer to the current object. In the following example, this refers to the newly created **Car** object, and assigns values to three properties:

```
function Car(color, make, model)
{
this.color = color;
this.make = make;
this.model = model;
}
```

For client versions of JScript, **this** refers to the **Window** object if used outside of the context of any other object.

JScript var Statement

Declares a variable.

Syntax: JScript var Statement

```
var variable [ = value ] [, variable2 [ = value2], ...]
```

Arguments: JScript var Statement

variable, *variable2*

The names of the variables being declared.

value, *value2*

The initial value assigned to the variable.

Remarks: JScript var Statement

Use the **var** statement to declare variables. These variables can be assigned values at declaration or later in your script. Examples of declaration follow:

```
var index;
var name = "Thomas Jefferson";
var answer = 42, counter, numpages = 10;
```

JScript with Statement

Establishes the default object for a statement.

Syntax: JScript with Statement

```
with (object)
statement
```

Arguments: JScript with Statement

object

The new default object.

statement

The statement for which *object* is the default object. Can be a compound statement.

Remarks: JScript with Statement

The **with** statement is commonly used to shorten the amount of code that you have to write in certain situations. In the example that follows, notice the repeated use of **Math**:

```
x = Math.cos(3 * Math.PI) + Math.sin(Math.LN10)
y = Math.tan(14 * Math.E)
```

When you use the **with** statement, your code becomes shorter and easier to read:

```
with (Math)
{
x = cos(3 * PI) + sin (LN10)
y = tan(14 * E)
}
```

JScript while Statement

Executes a statement until a specified condition is **False**.

Syntax: JScript while Statement

```
while (expression)
statement
```

Arguments: JScript while Statement

expression

A Boolean expression checked before each iteration of the loop. If *expression* is **True**, the loop is executed. If *expression* is **False**, the loop is terminated.

statement

The statement to be executed if *expression* is **True**. Can be a compound statement.

Remarks: JScript while Statement

The **while** statement checks *expression* before a loop is first executed. If *expression* is **False** at this time, the loop is never executed.

JScript Objects

JScript Array Object	Provides support for creation of arrays of any data type.
JScript Boolean Object	Creates a new Boolean value.
JScript Date Object	Enables basic storage and retrieval of dates and times.
JScript Dictionary Object	Object that stores data as key, item pairs.
JScript Drive Object	Provides access to the properties of a disk drive or network share.
JScript Enumerator Object	Enumerates the items in a collection.
JScript File Object	Provides access to the properties of a file.
JScript FileSystemObject Object	Provides access to a computer's file system.
JScript Folder Object	Provides access to the properties of a folder.
JScript Function Object	Creates a new function.
JScript Global Object	Intrinsic object that collects global methods into one object.
JScript Math Object	Intrinsic object that provides basic math functions.
JScript Number Object	An object representation of the number data type.
JScript Object	Provides functionality common to all JScript objects.
JScript RegExp Object	Stores information on regular expression pattern searches.
JScript Regular Expression Object	Contains a regular expression pattern.
JScript String Object	An object representation of the string data type.
JScript TextStream Object	Facilitates sequential access to file.
JScript VBArray Object	An object representation of VBScript safe arrays.
JScript Collections	Useful collections of other objects.

JScript Object

JScript Object

The JScript Object object provides functionality common to all JScript objects.

Methods: JScript Object

JScript Object valueOf Method	Returns the primitive value of the specified object.
JScript Object toString Method	Returns a string representation of an object.

Properties: JScript Object

JScript Object constructor Property	The function that creates an object.
JScript Object prototype Property	A reference to the prototype for a class of objects.

Syntax: JScript Object

```
new Object([value])
```

Arguments: JScript Object

value

Used if you want to convert a primitive data type (number, Boolean, string, or function) into an object. If omitted, an object with no contents is created. Optional.

Remarks: JScript Object

The **Object** object is contained in all other JScript objects—all of its methods and properties are available in all other objects. The methods can be redefined in objects you define, and are called by JScript at appropriate times. The **toString** method is an example of a frequently redefined **Object** method.

In this language reference, the description of each **Object** method includes both default and object-specific implementation information for the intrinsic JScript objects.

JScript Object constructor Property

Specifies the function that creates an object.

Syntax: JScript Object constructor Property

```
object.constructor
```

Arguments: JScript Object constructor Property

object

The name of an object or function.

Remarks: JScript Object constructor Property

The **constructor** property is a member of the prototype of every object that has a prototype. This includes all intrinsic JScript objects except the **Global** and **Math** objects. The **constructor** property contains a reference to the function that constructs instances of that particular object. For example:

```
x = new String("Hi");
if (x.constructor == String)
  // Do something (the condition will be true).
```

or

```
function MyFunc {
// Body of function.
}

y = new MyFunc;
if (y.constructor == MyFunc)
  // Do something (the condition will be true).
```

JScript Object prototype Property

Contains a reference to the prototype for a class of objects.

Syntax: JScript Object prototype Property

```
objectname.prototype
```

Arguments: JScript Object prototype Property

objectname

The name of an object.

Remarks: JScript Object prototype Property

Use the **prototype** property to provide a base set of functionality to a class of objects. New instances of an object "inherit" the behavior of the prototype assigned to that object.

For example, say you wanted to add a method to the **Array** object that returns the value of the largest element of the array. To do this, declare the function, add it to **Array.prototype**, and then use it.

```
function array_max( )
{
var i, max = this[0];
for (i = 1; i < this.length; i++)
{
```

```
if (max < this[i])
max = this[i];
}
return max;
}
Array.prototype.max = array_max;
var x = new Array(1, 2, 3, 4, 5, 6);
var y = x.max( );
```

After this code is executed, *y* contains the largest value in the array *x*, or 6.

All intrinsic JScript objects have a **prototype** property that is read-only. Functionality may be added to the prototype, as in the example, but the object may not be assigned a different prototype.

This is not the case for user-defined objects: User-defined objects may be assigned a new prototype.

The method and property lists for each intrinsic object in this language reference indicate which ones are part of the object's prototype, and which are not.

JScript Object toString Method

Returns a string representation of an object.

Syntax: JScript Object toString Method

```
objectname.toString( )
```

Arguments: JScript Object toString Method

objectname

An object for which a string representation is sought.

Remarks: JScript Object toString Method

The **toString** method is a member of all built-in JScript objects. How it behaves depends on the object type:

Object	Behavior
Array	Elements of an **Array** are converted to strings. The resulting strings are concatenated, separated by commas.
Boolean	If the Boolean value is **True**, returns "`true`". Otherwise, returns "`false`".
Function	Returns a string returned of the following form, where *functionname* is the name of the function whose **toString**

	method was called: `Function functionname( ) { [native code] }`
Number	Returns the textual representation of the number.
String	Returns the value of the **String** object.
Default	Returns "`[object objectname]`", where *objectname* is the name of the object type.

JScript Object valueOf Method

Returns the primitive value of the specified object.

Syntax: JScript Object valueOf Method

```
object.valueOf( )
```

Arguments: JScript Object valueOf Method

object

Any JScript object.

Remarks: JScript Object valueOf Method

The **valueOf** method is defined differently for each intrinsic JScript object.

Object	Return Value
Array	The elements of the array are converted into strings, and the strings are concatenated together, separated by commas.
Boolean	The Boolean value.
Date	The stored time value in milliseconds since midnight, January 1, 1970 UTC.
Function	The function itself.
Number	The numeric value.
Object	The object itself. This is the default.
String	The string value.

The **Math** object does not have a **valueOf** property.

JScript Array Object

JScript Array Object

The **Array** object provides support for creation of arrays of any data type.

Methods: JScript Array Object

JScript Array Object concat Method	Combines two arrays to make a new array.
JScript Array Object join Method	Converts all elements of an array into a **String** object and joins them.
JScript Array Object reverse Method	Reverses the elements of an array.
JScript Array Object slice Method	Returns a section of an array.
JScript Array Object sort Method	Sorts the elements of an array.

Properties: JScript Array Object

JScript Array Object length Property	An integer value one higher than the highest element defined in an array.

Syntax: JScript Array Object

```
new Array()

new Array(size)

new Array(element0, element1, ..., elementn)
```

Arguments: JScript Array Object

size

The size of the array. As arrays are zero-based, created elements will have indices from zero to size - 1.

element0,...,elementn

The elements to place in the array. This creates an array with *n* + 1 elements, and a length of *n*.

Remarks: JScript Array Object

After an array is created, the individual elements of the array can be accessed using [ ] notation, for example:

```
var my_array = new Array();

for (i = 0; i < 10; i++)

{

my_array[i] = i;

}

x = my_array[4];
```

Since arrays in Microsoft JScript are zero-based, the last statement in the preceding example accesses the fifth element of the array. That element contains the value 4.

If only one argument is passed to the **Array** constructor, and it is a number, it is coerced into an unsigned integer and the value is used as the size of the array. Otherwise, the parameter passed in is used as the only element of the array.

JScript Array Object concat Method

Combines two arrays to create a new array.

Syntax: JScript Array Object concat Method

```
array1.concat(array2)
```

Arguments: JScript Array Object concat Method

array1

An **Array** object to concatenate with *array2*. Required.

array2

An **Array** object to concatenate to the end of *array1*. Required.

Remarks: JScript Array Object concat Method

The **concat** method returns an **Array** object containing the concatenation of *array1* and *array2*.

If an object reference is copied from either *array1* or *array2* to the result, the object reference in the result still points to the same object. Changes to that object are reflected in both arrays.

JScript Array Object join Method

Converts all elements of an array into a **String** object and joins them.

Syntax: JScript Array Object join Method

```
arrayobj.join(separator)
```

Arguments: JScript Array Object join Method

arrayobj

The name of an **Array** object.

separator

A **String** object that is used to separate one element of an array from the next in the resulting **String** object. If omitted, the array elements are separated with an empty string.

Remarks: JScript Array Object join Method

The **join** method returns a **String** object that contains each element converted to a string and concatenated together. An example using the **join** method follows:

```
var my_array = new Array("Jan 5", 1996, "hey", "my birthday!");

var my_string = my_array.join(", ");
```

After execution, *my_string* contains

```
"Jan 5, 1996, hey, my birthday!"
```

JScript Array Object length Property

Specifies an integer value one higher than the highest element defined in an array.

Syntax: JScript Array Object length Property

```
numVar = arrayObj.length;
```

Remarks: JScript Array Object length Property

As the elements in an array do not have to be contiguous, the **length** property is not necessarily the number of elements in the array. For example, in the following array definition, `my_array.length` contains 7, not 2:

```
var my_array = new Array( );
my_array[0] = "Test";
my_array[6] = "Another Test";
```

If a value smaller than its previous value is assigned to the **length** property, the array is truncated and any elements with array indices equal to or greater than the new value of the **length** property are lost.

If a value larger than its previous value is assigned to the **length** property, the array is expanded, and any new elements created have the value undefined.

JScript Array Object reverse Method

Reverses the elements of an **Array** object.

Syntax: JScript Array Object reverse Method

```
arrayobj.reverse( )
```

Arguments: JScript Array Object reverse Method

arrayObj

The name of an **Array** object. Required.

Remarks: JScript Array Object reverse Method

The **reverse** method reverses the elements of an **Array** object in place. It does not create a new **Array** object during execution.

If the array is not contiguous, the **reverse** method creates elements in the array that fill the gaps in the array. Each of these created elements has the value undefined.

JScript Array Object slice Method

Returns a section of an array.

Syntax: JScript Array Object slice Method

```
arrayObj.slice(start, [end])
```

Arguments: JScript Array Object slice Method

arrayObj

An **Array** object. Required.

start

The zero-based index of the beginning of the specified portion of *arrayObj*. Required.

end

The zero-based index of the end of the specified portion of *arrayObj*. Optional.

Remarks: JScript Array Object slice Method

The slice method returns an **Array** object containing the specified portion of *arrayObj*.

The **slice** method copies up to, but not including, the element indicated by *end*. If negative, *end* indicates an offset from the end of *arrayObj*. In addition, it is not zero-based. If omitted, extraction continues to the end of *arrayObj*.

In the example that follows, all but the last element of *myArray* is copied into *newArray*:

```
newArray.slice(0, -1))
```

If an object reference is copied from *arrayObj* to the result, the object reference in the result still points to the same object. Changes to that object are reflected in both arrays.

JScript Array Object sort Method

Sorts the elements of an **Array** object.

Syntax: JScript Array Object sort Method

```
arrayObj.sort(sortfunction)
```

Arguments: JScript Array Object sort Method

arrayObj

The name of an **Array** object.

sortfunction

The name of the function used to determine the order of the elements. If omitted, the elements are sorted in ascending, ASCII-character order.

Remarks: JScript Array Object sort Method

The **sort** method sorts the **Array** object in place; no new **Array** object is created during execution.

If you supply a function in the *sortfunction* argument, it must return one of the following values:

- A negative value if the first argument passed is less than the second argument.
- Zero if the two arguments are equivalent.

- A positive value if the first argument is greater than the second argument.

JScript Boolean Object

JScript Boolean Object

The **Boolean** object creates a new Boolean value.

Methods & Properties: JScript Boolean Object

See **JScript Object** methods and properties.

Syntax: JScript Boolean Object

```
var variablename = new Boolean(boolvalue)
```

Arguments: JScript Boolean Object

boolvalue

The initital Boolean value for the new object. If this value is omitted, or is **False**, 0, **Null**, **NaN**, or an empty string, the initial value of the **Boolean** object is **False**. Otherwise, the initial value is **True**. Optional.

Remarks: JScript Boolean Object

The **Boolean** object is a wrapper for the Boolean data type. JScript implicitly uses the **Boolean** object whenever a Boolean data type is converted to a **Boolean** object.

You rarely call the **Boolean** object explicitly.

JScript Date Object

JScript Date Object

The **Date** object enables basic storage and retrieval of dates and times.

Methods: JScript Date Object

JScript Date Object getDate Method	Returns the day of the month according to local time.
JScript Date Object getDay Method	Returns the day of the week according to local time.
JScript Date Object getFullYear Method	Returns the 4-digit year according to local time.
JScript Date Object getHours Method	Returns the hours according to local time.
JScript Date Object getMilliseconds Method	Returns the number of milliseconds past the second according to local time.

JScript Date Object getMinutes Method	Returns the number of minutes past the hour according to local time.
JScript Date Object getMonth Method	Returns the month value according to local time.
JScript Date Object getSeconds Method	Returns the number of seconds past the minute according to local time.
JScript Date Object getTime Method	Returns the time stored in a **Date** object.
JScript Date Object getTimezoneOffset Method	Determines the difference in minutes between the time on the host computer and Universal Time Coordinated (UTC).
JScript Date Object getUTCDate Method	Returns the date of the month according to UTC.
JScript Date Object getUTCDay Method	Returns the day of the week according to UTC.
JScript Date Object getUTCFullYear Method	Returns the 4-digit year according to UTC.
JScript Date Object getUTCHours Method	Returns the hours according to UTC.
JScript Date Object getUTCMilliseconds Method	Returns the number of milliseconds past the second according to UTC.
JScript Date Object getUTCMinutes Method	Returns the number of minutes past the hour according to UTC.
JScript Date Object getUTCMonth Method	Returns the month according to UTC.
JScript Date Object getUTCSeconds Method	Returns the number of seconds past the minute according to UTC.
JScript Date Object getVarDate Method	Returns the VT_DATE value.
JScript Date Object getYear Method	Returns the 2-digit year.
JScript Date Object parse Method	Parses a string containing a date and returns the number of milliseconds between that date and midnight, January 1, 1970.
JScript Date Object setDate Method	Sets the numeric data according to local time.
JScript Date Object setFullYear Method	Sets the 4-digit year according to local time.
JScript Date Object setHours Method	Modifies the hours value according to local time.
JScript Date Object setMilliseconds Method	Modifies the milliseconds value according to local time.
JScript Date Object setMinutes Method	Modifies the minutes value according to local time.
JScript Date Object setMonth Method	Modifies the month according to local time.

JScript Date Object setSeconds Method	Modifies the seconds according to local time.
JScript Date Object setTime Method	Sets the date and time value directly.
JScript Date Object setUTCDate Method	Sets the numeric date in Universal Coordinated Time (UTC)
JScript Date Object setUTCFullYear Method	Sets the year value according to UTC.
JScript Date Object setUTCHours Method	Modifies the year value according to UTC.
JScript Date Object setUTCMilliseconds Method	Modifies the milliseconds according to UTC.
JScript Date Object setUTCMinutes Method	Modifies the minutes according to UTC.
JScript Date Object setUTCMonth Method	Modifies the month according to UTC.
JScript Date Object setUTCSeconds Method	Modifies the seconds according to UTC.
JScript Date Object setYear Method	Sets the 2-digit year.
JScript Date Object toGMTString Method	Converts the date to a string using GMT conventions.
JScript Date Object toLocaleString Method	Converts the date to a string using the current locale.
JScript Date Object toUTCString Method	Converts the date to a string using UTC conventions.
JScript Date Object UTC Method	Computes the number of milliseconds between midnight, January 1, 1970 Universal Coordinated Time (or GMT) and the supplied date.

Syntax: JScript Date Object

```
var newDateObj = new Date()

var newDateObj = new Date(dateVal)

var newDateObj = new Date(year, month, date[, hours[, minutes[,
seconds[,ms]]]])
```

Arguments: JScript Date Object

dateVal

If a numeric value, *dateVal* represents the number of milliseconds in Universal Coordinated Time between the specified date and midnight January 1, 1970. If a string, *dateVal* is parsed according to the rules in the **parse** method. The *dateVal* argument can also be a `VT_DATE` value as returned from some ActiveX objects.

year

The full year, for example, 1976 (and not 76). Required.

month

The month as an integer between 0 and 11 (January to December).

date

An integer between 1 and 31. Required.

hours

Must be supplied if *minutes* is supplied. An integer from 0 to 23 (midnight to 11pm) that specifies the hour. Optional.

minutes

Must be supplied if *seconds* is supplied. An integer from 0 to 59 that specifies the minutes. Optional.

seconds

Must be supplied if *milliseconds* is supplied. An integer from 0 to 59 that specifies the seconds. Optional.

ms

An integer from 0 to 999 that specifies the milliseconds. Optional.

Remarks: JScript Date Object

A **Date** object contains a number representing a particular instant in time to within a millisecond. If the value of an argument is greater than its range or is a negative number, other stored values are modified accordingly. For example, if you specify 150 seconds, JScript redefines that number as 2 minutes and 30 seconds.

If the number is **NaN**, that indicates that the object does not represent a specific instant of time. If you pass no parameters to the **Date** object, it is initialized to the current time (UTC). A value must be given to the object before you can use it.

The range of dates that can be represented in a **Date** object is approximately 285,616 years on either side of January 1, 1970.

The **Date** object has two static methods that are called without creating a **Date** object. They are the **parse** and **UTC** methods.

JScript Date Object getDate Method

Returns the day of the month as stored in a **Date** object according to local time.

Syntax: JScript Date Object getDate Method

```
objDate.getDate()
```

Arguments: JScript Date Object getDate Method

objDate

The name of a **Date** object. Required.

Remarks: JScript Date Object getDate Method

To get the date value according to Universal Coordinated Time (UTC), use the **getUTCDate** method.

The return value is an integer between 1 and 31 that represents the date stored in the **Date** object.

JScript Date Object getDay Method

Retrieves the day of the week represented by the date stored in a **Date** object according to local time.

Syntax: JScript Date Object getDay Method

```
objDate.getDay()
```

Arguments: JScript Date Object getDay Method

objDate

The name of a **Date** object. Required.

Remarks: JScript Date Object getDay Method

To get the day according to Universal Coordinated Time (UTC), use the **getUTCDay** method.

The integer returned from the **getDay** method is an integer between 0 and 6 representing the day of the week and corresponds to a day of the week as follows:

0 = Sunday

1 = Monday

2 = Tuesday

3 = Wednesday

4 = Thursday

5 = Friday

6 = Saturday

JScript Date Object getFullYear Method

Returns the year stored in the **Date** object according to local time.

Syntax: JScript Date Object getFullYear Method

```
objDate.getFullYear()
```

Arguments: JScript Date Object getFullYear Method

objDate

The name of a **Date** object.

Remarks: JScript Date Object getFullYear Method

To get the year according to Universal Coordinated Time (UTC), use the **getUTCFullYear** method.

The **getFullYear** method returns the year as an absolute number. For example, the year 1976 is returned as 1976. This avoids problems with dates occurring at the end of the 20th century.

JScript Date Object getHours Method

Retrieves the hours stored in a **Date** object according to local time.

Syntax: JScript Date Object getHours Method

```
objDate.getHours()
```

Arguments: JScript Date Object getHours Method

objDate

The name of a **Date** object.

Remarks: JScript Date Object getHours Method

To get the hours value according to Universal Coordinated Time (UTC), use the **getUTCHours** method.

The **getHours** method returns an integer between 0 and 23 indicating the number of hours since midnight. A zero occurs in two situations: the time is before 1:00:00 am, or the time was not stored in the **Date** object when the object was created. The only way to determine which situation you have is to also check the minutes and seconds for zero values. If they are all zeroes, it is nearly certain that the time was not stored in the **Date** object.

JScript Date Object getMilliseconds Method

Retrieves the number of milliseconds past the second from the milliseconds value stored in a **Date** object according to local time.

Syntax: JScript Date Object getMilliseconds Method

```
objDate.getMilliseconds()
```

Arguments: JScript Date Object getMilliseconds Method

objDate

The name of a **Date** object. Required.

Remarks: JScript Date Object getMilliseconds Method

To get the number of milliseconds in Universal Coordinated Time (UTC), use the **getUTCMilliseconds** method.

The millisecond value returned can range from 0-999.

JScript Date Object getMinutes Method

Retrieves the number of minutes past the hour from the minutes value stored in a **Date** object according to local time.

Syntax: JScript Date Object getMinutes Method

```
objDate.getMinutes()
```

Arguments: JScript Date Object getMinutes Method

objDate

The name of a **Date** object. Required.

Remarks: JScript Date Object getMinutes Method

To get the minutes value according to Universal Coordinated Time (UTC), use the **getUTCMinutes** method.

The **getMinutes** method returns an integer between 0 and 59 equal to the minutes value stored in the **Date** object. A zero is returned in two situations: one occurs when the time is less than one minute after the hour. The other occurs when the time was not stored in the **Date** object when the object was created. The only way to determine which situation you have is to also check the hours and seconds for zero values. If they are all zeroes, it is nearly certain that the time was not stored in the **Date** object.

JScript Date Object getMonth Method

Retrieves the month value of the **Date** object according to local time.

Syntax: JScript Date Object getMonth Method

```
objDate.getMonth()
```

Arguments: JScript Date Object getMonth Method

objDate

The name of a **Date** object. Required.

Remarks: JScript Date Object getMonth Method

To get the month value according to Universal Coordinated Time (UTC), use the **getUTCMonth** method.

The **getMonth** method returns an integer between 0 and 11 indicating the month stored in the **Date** object. The integer returned is not the traditional number used to indicate the month—it is one less. If "Jan 5, 1996 08:47:00" is stored in a **Date** object, **getMonth** returns 0.

JScript Date Object getSeconds Method

Retrieves the number of seconds past the minute from the seconds value stored in a **Date** object according to local time.

Syntax: JScript Date Object getSeconds Method

```
objDate.getSeconds()
```

Arguments: JScript Date Object getSeconds Method

objDate

The name of a **Date** object. Required.

Remarks: JScript Date Object getSeconds Method

To get the seconds value according to Universal Coordinated Time (UTC), use the **getUTCSeconds** method.

The **getSeconds** method returns an integer between 0 and 59 indicating the seconds value of the indicated **Date** object. A zero is returned in two situations. One occurs when the time is less than one second into the current minute. The other occurs when the time was not stored in the **Date** object when the object was created. The only way to determine which situation you have is to also check the hours and minutes for zero values. If they are all zeroes, it is nearly certain that the time was not stored in the **Date** object.

JScript Date Object getTime Method

Retrieves the time stored in a **Date** object.

Syntax: JScript Date Object getTime Method

```
objDate.getTime()
```

Arguments: JScript Date Object getTime Method

objDate

The name of a **Date** object. Required.

Remarks: JScript Date Object getTime Method

The **getTime** method returns an integer value representing the number of milliseconds between midnight, January 1, 1970 and the time stored in the **Date** object. The range of dates is approximately 285,616 years from either side of midnight, January 1, 1970. Negative numbers indicate dates prior to 1970.

When doing multiple date and time calculations, it is frequently useful to define variables equal to the number of milliseconds in a day, hour, or minute. For example:

```
varMinuteMilli = 1000 * 60

varHourMilli = varMinuteMilli * 60

varDayMilli = varHourMilli * 24
```

JScript Date Object getTimezoneOffset Method

Determines the difference in minutes between the time on the host computer and Universal Coordinated Time (UTC).

Syntax: JScript Date Object getTimezoneOffset Method

```
objDate.getTimezoneOffset()
```

Arguments: JScript Date Object getTimezoneOffset Method

objDate

The name of a **Date** object. Required.

Remarks: JScript Date Object getTimezoneOffset Method

The **getTimezoneOffset** method returns an integer value representing the number of minutes between the time on the current machine and UTC. These values are appropriate to the computer the script is executed on. If it is called from a server script, the return value is appropriate to the server. If it is called from a client script, the return value is appropriate to the client.

This number will be positive if you are behind UTC (e.g., Pacific Daylight Time), and negative if you are ahead of UTC (e.g., Japan).

For example, suppose a server in New York City is contacted by a client in Los Angeles on December 1. **getTimezoneOffset** returns 480 if executed on the client, or 300 if executed on the server.

JScript Date Object getUTCDate Method

Returns the date of the month stored in a **Date** object according to Universal Coordinated Time (UTC).

Syntax: JScript Date Object getUTCDate Method

```
objDate.getUTCDate()
```

Arguments: JScript Date Object getUTCDate Method

objDate

The name of a **Date** object. Required.

Remarks: JScript Date Object getUTCDate Method

To get the date according to local time, use the **getDate** method.

The return value is an integer between 1 and 31 that represents the date stored in the **Date** object.

JScript Date Object getUTCDay Method

Returns the day of the week as stored in a **Date** object according to Universal Coordinated Time (UTC).

Syntax: JScript Date Object getUTCDay Method

```
objDate.getUTCDay()
```

Arguments: JScript Date Object getUTCDay Method

objDate

The name of a **Date** object. Required.

Remarks: JScript Date Object getUTCDay Method

To get the day of the week according to local time, use the **getDay** method.

The value returned by the **getUTCDay** method is an integer between 0 and 6 representing the day of the week and corresponds to a day of the week as follows:

0 = Sunday

1 = Monday

2 = Tuesday

3 = Wednesday

4 = Thursday

5 = Friday

6 = Saturday

JScript Date Object getUTCFullYear Method

Returns the year stored in a **Date** object according to Universal Coordinated Time (UTC).

Syntax: JScript Date Object getUTCFullYear Method

```
objDate.getUTCFullYear()
```

Arguments: JScript Date Object getUTCFullYear Method

objDate

The name of a **Date** object. Required.

Remarks: JScript Date Object getUTCFullYear Method

To get the year according to local time, use the **getFullYear** method.

The **getUTCFullYear** method returns the year as an absolute number. This avoids problems with dates occurring at the end of the 20th century.

JScript Date Object getUTCHours Method

Returns the hours stored in a **Date** object according to Universal Coordinated Time (UTC).

Syntax: JScript Date Object getUTCHours Method

```
objDate.getUTCHours()
```

Arguments: JScript Date Object getUTCHours Method

objDate

The name of a **Date** object. Required.

Remarks: JScript Date Object getUTCHours Method

To get the number of hours elapsed since midnight using local time, use the **getHours** method.

The **getUTCHours** method returns an integer between 0 and 23 indicating the number of hours since midnight. A zero occurs in two situations: the time is before 1:00:00 A.M., or a time was not stored in the **Date** object when the object was created. The only way to determine which situation you have is to also check the minutes and seconds for zero values. If they are all zeroes, it is nearly certain that the time was not stored in the **Date** object.

JScript Date Object getUTCMilliseconds Method

Retrieves the number of milliseconds past the second from the milliseconds value stored in a **Date** object according to Universal Coordinated Time (UTC).

Syntax: JScript Date Object getUTCMilliseconds Method

```
objDate.getUTCMilliseconds()
```

Arguments: JScript Date Object getUTCMilliseconds Method

objDate

The name of a **Date** object. Required.

Remarks: JScript Date Object getUTCMilliseconds Method

To get the number of milliseconds in local time, use the **getMilliseconds** method.

The millisecond value returned can range from 0-999.

JScript Date Object getUTCMinutes Method

Retrieves the number of minutes past the hour from the minutes value stored in a **Date** object according to Universal Coordinated Time (UTC).

Syntax: JScript Date Object getUTCMinutes Method

```
objDate.getUTCMinutes()
```

Arguments: JScript Date Object getUTCMinutes Method

objDate

The name of a **Date** object. Required.

Remarks: JScript Date Object getUTCMinutes Method

To get the number of minutes stored using local time, use the **getMinutes** method.

The **getUTCMinutes** method returns an integer between 0 and 59 equal to the number of minutes stored in the **Date** object. A zero occurs in two situations: the time is less than one minute after the hour, or a time was not stored in the **Date** object when the object was created. The only way to determine which situation you have is to also check the hours and seconds for zero values. If they are all zeroes, it is nearly certain that the time was not stored in the **Date** object.

JScript Date Object getUTCMonth Method

Retrieves the month value stored in a **Date** object according to Universal Coordinated Time (UTC).

Syntax: JScript Date Object getUTCMonth Method

```
objDate.getUTCMonth()
```

Arguments: JScript Date Object getUTCMonth Method

objDate

The name of a **Date** object. Required.

Remarks: JScript Date Object getUTCMonth Method

To get the month in local time, use the **getMonth** method.

The **getUTCMonth** method returns an integer between 0 and 11 indicating the month stored in the **Date** object. The integer returned is not the traditional number used to indicate the month—it is one less. If "Jan 5, 1996 08:47:00.0" is stored in a **Date** object, **getUTCMonth** returns 0.

JScript Date Object getUTCSeconds Method

Retrieves the number of seconds past the minute from the seconds value stored in a **Date** object according to Universal Coordinated Time (UTC).

Syntax: JScript Date Object getUTCSeconds Method

```
objDate.getUTCSeconds()
```

Arguments: JScript Date Object getUTCSeconds Method

objDate

The name of a **Date** object. Required.

Remarks: JScript Date Object getUTCSeconds Method

To get the number of seconds in local time, use the **getSeconds** method.

The **getUTCSeconds** method returns an integer between 0 and 59 indicating the seconds value of the indicated **Date** object. A zero occurs in two situations: the time is less than one second into the current minute, or a time was not stored in the **Date** object when the object was created. The only way to determine which situation you have is to also check the minutes and hours for zero values. If they are all zeroes, it is nearly certain that the time was not stored in the **Date** object.

JScript Date Object getVarDate Method

Returns the `VT_DATE` value stored in the **Date** object.

Syntax: JScript Date Object getVarDate Method

```
objDate.getVarDate( )
```

Arguments: JScript Date Object getVarDate Method

objDate

The name of a **Date** object. Required.

Remarks: JScript Date Object getVarDate Method

The **getVarDate** method is used when interacting with ActiveX or other objects that accept and return date values in `VT_DATE` format.

JScript Date Object getYear Method

Retrieves the year stored in the specified **Date** object.

Syntax: JScript Date Object getYear Method

```
objDate.getYear()
```

Arguments: JScript Date Object getYear Method

objDate

The name of a **Date** object. Required.

Remarks: JScript Date Object getYear Method

This method is obsolete, and is provided for backwards compatibility only. Use the **getFullYear** method instead.

The year is an integer value and is returned as the difference between the stored year and 1900. For example, 1996 is returned as 96, and 2025 is returned as 125.

JScript Date Object parse Method

Parses a string containing a date, and returns the number of milliseconds between that date and midnight, January 1, 1970.

Syntax: JScript Date Object parse Method

```
Date.parse(dateVal)
```

Arguments: JScript Date Object parse Method

dateVal

The required *dateVal* argument is either a string containing a date in a format such as "Jan 5, 1996 08:47:00" or a `VT_DATE` value retrieved from an ActiveX object or other object.

Remarks: JScript Date Object parse Method

The **parse** method returns an integer value representing the number of milliseconds between midnight, January 1, 1970 and the date supplied in *dateVal*.

The **parse** method is a static method of the **Date** object. Because it is a static method, it is invoked as shown in the following code rather than invoked as a method of a created **Date** object.

```
var datestring = "November 1, 1997 10:15 AM";

Date.parse(datestring)
```

The following rules govern what the **parse** method can successfully parse:

- Short dates can use either a "/" or "-" date separator, but must follow the month/day/year format, for example "7/20/96."
- Long dates of the form "July 10 1995" can be given with the year, month, and day in any order, and the year in 2- or 4-digit form. If you use the 2-digit form, the year must be greater than or equal to 70.
- Any text inside parentheses is treated as a comment. These parentheses may be nested.
- Both commas and spaces are treated as delimiters. Multiple delimiters are permitted. Month and day names must have two or more characters. Two character names that are not unique are resolved as the last match. For example, "Ju" is resolved as July, not June.
- The stated day of the week is ignored if it is incorrect given the remainder of the supplied date. For example, "Tuesday November 9 1996" is accepted and parsed even though that date actually falls on a Friday. The resulting **Date** object contains "Friday November 9 1996."

Note

JScript handles all standard time zones, as well as Universal Coordinated Time (UTC) and Greenwich Mean Time (GMT). Hours, minutes, and seconds are separated by colons, although all need not be specified. "10:", "10:11", and "10:11:12" are all valid. If the 24-hour clock is used, it is an error to specify "PM" for times later than 12 noon. For example, "23:15 PM" is an error. A string containing an invalid date is an error. For example, a string containing two years or two months is an error.

JScript Date Object setDate Method

Sets the numeric date of the **Date** object according to local time.

Syntax: JScript Date Object setDate Method

```
objDate.setDate(numDate)
```

Arguments: JScript Date Object setDate Method

objDate

The name of a **Date** object. Required.

numDate

A numeric value equal to the numeric date.

Remarks: JScript Date Object setDate Method

To set the date value according to Universal Coordinated Time (UTC), use the **setUTCDate** method.

If the value of *numDate* is greater than the number of days in the month stored in the **Date** object or is a negative number, the date is set to a date equal to *numDate* minus the number of days in the stored month. For example, if the stored date is January 5, 1996, and setDate(32) is called, the date changes to February 1, 1996. Negative numbers have a similar behavior.

JScript Date Object setFullYear Method

Sets the year value in the **Date** object according to local time.

Syntax: JScript Date Object setFullYear Method

```
objDate.setFullYear(numYear[, numMonth[, numDate]])
```

Arguments: JScript Date Object setFullYear Method

objDate

The name of a **Date** object. Required.

numYear

A numeric value equal to the year. Required.

numMonth

A numeric value equal to the month. Must be supplied if *numDate* is supplied. Optional.

numDate

A numeric value equal to the date. Optional.

Remarks: JScript Date Object setFullYear Method

All **set** methods taking optional arguments use the value returned from corresponding **get** methods, if you do not specify an optional argument. For example, if the *numMonth* argument is optional, but not specified, JScript uses the value returned from the **getMonth** method.

In addition, if the value of an argument is greater that its range or is a negative number, other stored values are modified accordingly.

To set the year according to Universal Coordinated Time (UTC), use the **setUTCFullYear** method.

The range of years supported in the date object is approximately 285,616 years from either side of 1970.

JScript Date Object setHours Method

Modifies the hours stored in the **Date** object according to local time.

Syntax: JScript Date Object setHours Method

```
objDate.setHours(numHours[, numMin[, numSec[, numMilli]]])
```

Arguments: JScript Date Object setHours Method

objDate

The name of a **Date** object. Required.

numHours

A numeric value equal to the hours value. Required.

numMin

A numeric value equal to the minutes value. Must be supplied if either of the following arguments are used. Optional.

numSec

A numeric value equal to the seconds value. Must be supplied if the following argument is used. Optional.

numMilli

A numeric value equal to the milliseconds value. Optional.

Remarks: JScript Date Object setHours Method

All **set** methods taking optional arguments use the value returned from corresponding **get** methods, if you do not specify an optional argument. For example, if the *numMin* argument is optional, but not specified, JScript uses the value returned from the **getMinutes** method.

To set the hours value according to Universal Coordinated Time (UTC), use the **setUTCHours** method.

If the value of an argument is greater than its range or is a negative number, other stored values are modified accordingly. For example, if the stored date is "Jan 5, 1996 00:00:00," and setHours(30) is called, the date is changed to "Jan 6, 1996 06:00:00." Negative numbers have a similar behavior.

JScript Date Object setMilliseconds Method

Modifies the milliseconds value stored in the **Date** object according to local time.

Syntax: JScript Date Object setMilliseconds Method

```
objDate.setMilliseconds(numMilli)
```

Arguments: JScript Date Object setMilliseconds Method

objDate

The name of a **Date** object. Required.

numMilli

A numeric value equal to the millisecond value.

Remarks: JScript Date Object setMilliseconds Method

To set the milliseconds value according to Universal Coordinated Time (UTC), use the **setUTCMilliseconds** method.

If the value of *numMilli* is greater than 999 or is a negative number, the stored number of seconds (and minutes, hours, and so forth if necessary) is incremented an appropriate amount.

JScript Date Object setMinutes Method

Modifies the minutes stored in the **Date** object according to local time.

Syntax: JScript Date Object setMinutes Method

```
objDate.setMinutes(numMinutes[, numSeconds[, numMilli]])
```

Arguments: JScript Date Object setMinutes Method

objDate

The name of a **Date** object. Required.

numMinutes

A numeric value equal to the minutes value. Required.

numSeconds

A numeric value equal to the seconds value. Must be supplied if the *numMilli* argument is used. Optional.

numMilli

A numeric value equal to the milliseconds value. Optional.

Remarks: JScript Date Object setMinutes Method

All **set** methods taking optional arguments use the value returned from corresponding **get** methods, if you do not specify an optional argument. For example, if the *numSeconds* argument is optional, but not specified, JScript uses the value returned from the **getSeconds** method.

To set the minutes value according to Universal Coordinated Time (UTC), use the **setUTCMinutes** method.

If the value of an argument is greater than its range or is a negative number, other stored values are modified accordingly. For example, if the stored date is "Jan 5, 1996 00:00:00" and setMinutes(90) is called, the date is changed to "Jan 5, 1996 01:30:00." Negative numbers have a similar behavior.

JScript Date Object setMonth Method

Modifies the month stored in the **Date** object according to local time.

Syntax: JScript Date Object setMonth Method

```
objDate.setMonth(numMonth[, dateVal])
```

Arguments: JScript Date Object setMonth Method

objDate

The name of a **Date** object. Required.

numMonth

A numeric value equal to the month. Required.

dateVal

A numeric value representing the date. If not supplied, the value from a call to the **getDate** method is used. Optional.

Remarks: JScript Date Object setMonth Method

To set the month value according to Universal Coordinated Time (UTC), use the **setUTCMonth** method.

If the value of *numMonth* is greater than 11 (January is month 0) or is a negative number, the stored year is modified accordingly. For example, if the stored date is "Jan 5, 1996" and **setMonth(14)** is called, the date is changed to "Mar 5, 1997."

JScript Date Object setSeconds Method

Modifies the seconds value stored in the **Date** object according to local time.

Syntax: JScript Date Object setSeconds Method

```
objDate.setSeconds(numSeconds[, numMilli])
```

Arguments: JScript Date Object setSeconds Method

objDate

The name of a **Date** object. Required.

numSeconds

A numeric value equal to the seconds value. Required.

numMilli

A numeric value equal to the milliseconds value. Optional.

Remarks: JScript Date Object setSeconds Method

All **set** methods taking optional arguments use the value returned from corresponding **get** methods, if you do not specify an optional argument. For example, if the *numMilli* argument is optional, but not specified, JScript uses the value returned from the **getMilliseconds** method.

To set the seconds value according to Universal Coordinated Time (UTC), use the **setUTCSeconds** method.

If the value of an argument is greater than its range or is a negative number, other stored values are modified accordingly. For example, if the stored date is "Jan 5, 1996 00:00:00" and **setSeconds(150)** is called, the date is changed to "Jan 5, 1996 00:02:30."

JScript Date Object setUTCDate Method

Sets the numeric date of the **Date** object in Universal Coordinated Time (UTC).

Syntax: JScript Date Object setUTCDate Method

```
objDate.setUTCDate(numDate)
```

Arguments: JScript Date Object setUTCDate Method

objDate

The name of a **Date** object. Required.

numDate

A numeric value equal to the numeric date.

Remarks: JScript Date Object setUTCDate Method

To set the date according to local time, use the **setDate** method.

If the value of *numDate* is greater than the number of days in the month stored in the **Date** object or is a negative number, the date is set to a date equal to *numDate* minus the number of days in the stored month. For example if the stored date is January 5, 1996, and **setUTCDate(32)** is called, the date changes to February 1, 1996. Negative numbers have a similar behavior.

JScript Date Object setTime Method

Sets the date and time value directly in the **Date** object.

Syntax: JScript Date Object setTime Method

```
objDate.setTime(milliseconds)
```

Arguments: JScript Date Object setTime Method

objDate

The name of a **Date** object. Required.

milliseconds

An integer value representing the number of elapsed milliseconds since midnight, January 1, 1970 GMT.

Remarks: JScript Date Object setTime Method

If *milliseconds* is negative, it indicates a date before 1970. The range of available dates is approximately 285,616 years from either side of 1970.

Setting the date and time with the **setTime** method is independent of the time zone.

JScript Date Object setUTCFullYear Method

Sets the year value in the **Date** object according to Universal Coordinated Time (UTC).

Syntax: JScript Date Object setUTCFullYear Method

```
objDate.setUTCFullYear(numYear[, numMonth[, numDate]])
```

Arguments: JScript Date Object setUTCFullYear Method

objDate

The name of a **Date** object. Required.

numYear

A numeric value equal to the year. Required.

numMonth

A numeric value equal to the month. Must be supplied if *numDate* is supplied. Optional.

numDate

A numeric value equal to the date. Optional.

Remarks: JScript Date Object setUTCFullYear Method

All **set** methods taking optional arguments use the value returned from corresponding **get** methods, if you do not specify an optional argument. For example, if the *numMonth* argument is optional, but not specified, JScript uses the value returned from the **getUTCMonth** method.

In addition, if the value of an argument is greater that its range or is a negative number, other stored values are modified accordingly.

To set the year according to local time, use the **setFullYear** method.

The range of years supported in the **Date** object is approximately 285,616 years from either side of 1970.

JScript Date Object setUTCHours Method

Modifies the hours stored in the **Date** object according to Universal Coordinated Time (UTC).

Syntax: JScript Date Object setUTCHours Method

```
objDate.setUTCHours(numHours[, numMin[, numSec[, numMilli]]])
```

Arguments: JScript Date Object setUTCHours Method

objDate

The name of a **Date** object. Required.

numHours

A numeric value equal to the hours value. Required.

numMin

A numeric value equal to the minutes value. Must be supplied if either *numSec* or *numMilli* are used. Optional.

numSec

A numeric value equal to the seconds value. Must be supplied if *numMilli* argument is used. Optional.

numMilli

Optional. A numeric value equal to the milliseconds value.

Remarks: JScript Date Object setUTCHours Method

All **set** methods taking optional arguments use the value returned from corresponding **get** methods, if you do not specify an optional argument. For example, if the *numHours* argument is optional, but not specified, JScript uses the value returned from the **getUTCHours** method.

To set the hours value according to local time, use the **setHours** method.

If the value of an argument is greater than its range or is a negative number, other stored values are modified accordingly. For example, if the stored date is "Jan 5, 1996 00:00:00.00," and **setUTCHours(30)** is called, the date is changed to "Jan 6, 1996 06:00:00.00."

JScript Date Object setUTCMilliseconds Method

Modifies the milliseconds value stored in the **Date** object according to Universal Coordinated Time (UTC).

Syntax: JScript Date Object setUTCMilliseconds Method

```
objDate.setUTCMilliseconds(numMilli)
```

Arguments: JScript Date Object setUTCMilliseconds Method

objDate

The name of a **Date** object. Required.

numMilli

A numeric value equal to the milliseconds value.

Remarks: JScript Date Object setUTCMilliseconds Method

To set the milliseconds according to local time, use the **setMilliseconds** method.

If the value of *numMilli* is greater than 999 or is a negative number, the stored number of seconds (and minutes, hours, and so forth if necessary) is incremented an appropriate amount.

JScript Date Object setUTCMinutes Method

Modifies the minutes value of the **Date** object according to Universal Coordinated Time (UTC).

Syntax: JScript Date Object setUTCMinutes Method

```
objDate.setUTCMinutes(numMinutes[, numSeconds[, numMilli]])
```

Arguments: JScript Date Object setUTCMinutes Method

objDate

The name of a **Date** object. Required.

numMinutes

A numeric value equal to the minutes value. Required.

numSeconds

A numeric value equal to the seconds value. Must be supplied if *numMilli* is used. Optional.

numMilli

A numeric value equal to the milliseconds value. Optional.

Remarks: JScript Date Object setUTCMinutes Method

All **set** methods taking optional arguments use the value returned from corresponding **get** methods, if you do not specify an optional argument. For example, if the *numSeconds* argument is optional, but not specified, JScript uses the value returned from the **getUTCSeconds** method.

To modify the minutes value according to local time, use the **setMinutes** method.

If the value of an argument is greater than its range or is a negative number, other stored values are modified accordingly. For example, if the stored date is "Jan 5, 1996 00:00:00.00," and **setUTCMinutes(70)** is called, the date is changed to "Jan 5, 1996 01:10:00.00."

JScript Date Object setUTCMonth Method

Modifies the month stored in the **Date** object according to Universal Coordinated Time (UTC).

Syntax: JScript Date Object setUTCMonth Method

```
objDate.setUTCMonth(numMonth[, dateVal])
```

Arguments: JScript Date Object setUTCMonth Method

objDate

The name of a **Date** object. Required.

numMonth

A numeric value equal to the month. Required.

dateVal

A numeric value representing the date. If not supplied, the value from a call to the **getUTCDate** method is used. Optional.

Remarks: JScript Date Object setUTCMonth Method

To set the month value according to local time, use the **setMonth** method.

If the value of *numMonth* is greater than 11 (January is month 0) or is a negative number, the stored year is incremented or decremented appropriately.

For example, if the stored date is "Jan 5, 1996 00:00:00.00," and **setUTCMonth(14)** is called, the date is changed to "Mar 5, 1997 00:00:00.00."

JScript Date Object setUTCSeconds Method

Modifies the seconds value stored in the **Date** object according to Universal Coordinated Time (UTC).

Syntax: JScript Date Object setUTCSeconds Method

```
objDate.setUTCSeconds(numSeconds[, numMilli])
```

Arguments: JScript Date Object setUTCSeconds Method

objDate

The name of a **Date** object. Required.

numSeconds

A numeric value equal to the seconds value. Required.

numMilli

A numeric value equal to the milliseconds value. Optional.

Remarks: JScript Date Object setUTCSeconds Method

All **set** methods taking optional arguments use the value returned from corresponding **get** methods, if you do not specify an optional argument. For example, if the *numMilli* argument is optional, but not specified, JScript uses the value returned from the **getUTCMilliseconds** method.

To set the seconds value according to local time, use the **setSeconds** method.

If the value of an argument is greater than its range or is a negative number, other stored values are modified accordingly. For example, if the stored date is "Jan 5, 1996 00:00:00.00" and **setSeconds(150)** is called, the date is changed to "Jan 5, 1996 00:02:30.00."

JScript Date Object setYear Method

Sets the year value in the **Date** object.

Syntax: JScript Date Object setYear Method

```
objDate.setYear(numYear)
```

Arguments: JScript Date Object setYear Method

objDate

The name of a **Date** object. Required.

numYear

A numeric value equal to the year minus 1900.

Remarks: JScript Date Object setYear Method

This method is obsolete, and is maintained for backwards compatibility only. Use the **setFullYear** method instead.

To set the year of a **Date** object to 1997, call **setYear(97)**. To set the year to 110, call **setYear(110)**. Finally, to set the year to a year in the range 0-99, use the **setFullYear** method.

JScript Date Object toGMTString Method

Converts the date to a string using GMT convention.

Syntax: JScript Date Object toGMTString Method

```
objDate.toGMTString()
```

Arguments: JScript Date Object toGMTString Method

objDate

The name of a **Date** object. Required.

Remarks: JScript Date Object toGMTString Method

The **toGMTString** method is obsolete, and is provided for backwards compatibility only. It is recommended that you use the **toUTCString** method instead

The **toGMTString** method returns a **String** object that contains the date formatted using GMT convention. The format of the return value is as follows: "05 Jan 1996 00:00:00 GMT."

JScript Date Object toLocaleString Method

Converts the date to a string using the current locale.

Syntax: JScript Date Object toLocaleString Method

```
dateObj.toLocaleString( )
```

Arguments: JScript Date Object toLocaleString Method

objDate

The name of a **Date** object. Required.

Remarks: JScript Date Object toLocaleString Method

The **toLocaleString** method returns a **String** object that contains the date written in the current locale's default format. The format of the return value depends on the current locale. For example, in the United States, **toLocaleString** may return "01/05/96 00:00:00" for January 5, but in Europe, it may return "05/01/96 00:00:00" for the same date, as European convention puts the day before the month.

JScript Date Object toUTCString Method

Converts the date to a string in Universal Coordinated Time (UTC).

Syntax: JScript Date Object toUTCString Method

```
objDate.toUTCString()
```

Arguments: JScript Date Object toUTCString Method

objDate

The name of a **Date** object. Required.

Remarks: JScript Date Object toUTCString Method

The **toUTCString** method returns a **String** object that contains the date formatted using UTC convention in a convenient, easily readable form.

JScript Date Object UTC Method

Computes the number of milliseconds between midnight, January 1, 1970 Universal Coordinated Time (UTC) (or GMT) and the supplied date.

Syntax: JScript Date Object UTC Method

```
Date.UTC(year, month, day[, hours[, minutes[, seconds[,ms]]]])
```

Arguments: JScript Date Object UTC Method

year

The full year designation is required for cross-century date accuracy. If *year* between 0 and 99 is used, then *year* is assumed to be 1900 + *year*. Required.

month

The month as an integer between 0 and 11 (January to December). Required.

date

The date as an integer between 1 and 31. Required.

hours

Must be supplied if *minutes* is supplied. An integer from 0 to 23 (midnight to 11pm) that specifies the hour. Optional.

minutes

Must be supplied if *seconds* is supplied. An integer from 0 to 59 that specifies the minutes. Optional.

seconds

Must be supplied if *milliseconds* is supplied. An integer from 0 to 59 that specifies the seconds. Optional.

ms

An integer from 0 to 999 that specifies the milliseconds. Optional.

Remarks: JScript Date Object UTC Method

The **UTC** method returns the number of milliseconds between midnight, January 1, 1970 UTC and the supplied date. This return value can be used in the **setTime** method and in the **Date** object constructor. If the value of an argument is greater than its range or is a negative number, other stored values are modified accordingly. For example, if you specify 150 seconds, JScript redefines that number as two minutes and 30 seconds.

The difference between the **UTC** method and the **Date** object constructor that accepts a date is that the **UTC** method assumes UTC, and the **Date** object constructor assumes local time.

The **UTC** method is a static method. Therefore, a **Date** object does not have to be created before it can be used. The **UTC** method is invoked as follows:

```
var datestring = "November 1, 1997 10:15 AM";
Date.UTC(datestring)
```

Note

If *year* is between 0 and 99, use *1900* + *year* for the year.

JScript Dictionary Object

JScript Dictionary Object

The **Dictionary** object stores data key, item pairs.

Methods: JScript Dictionary Object

JScript Dictionary Object Add Method	Adds a key, item pair to a **Dictionary** object.
JScript Dictionary Object Exists Method	Returns a Boolean value indicating the existence of a key.
JScript Dictionary Object Items Method	Returns an array containing all the existing items in a **Dictionary** object..
JScript Dictionary Object Keys Method	Returns an array containing all the existing keys in a dictionary object.

JScript Dictionary Object Remove Method	Removes a key, item pair.
JScript Dictionary Object RemoveAll Method	Removes all key, item pairs.

Properties: JScript Dictionary Object

JScript Dictionary Object Key Property	A key in a **Dictionary** object.
JScript Dictionary Object Count Property	The number of items in a **Dictionary** object.
JScript Dictionary Object Item Property	An item associated with a key in a **Dictionary** object.

Syntax: JScript Dictionary Object

```
y = new ActiveXObject("Scripting.Dictionary")
```

Remarks: JScript Dictionary Object

A **Dictionary** object is the equivalent of a PERL associative array. Items, which can be any form of data, are stored in the array. Each item is associated with a unique key. The key is used to retrieve an individual item and is usually an integer or a string, but can be anything except an array.

The following code illustrates how to create a **Dictionary** object:

```
var y = new ActiveXObject("Scripting.Dictionary");
y.add ("a", "test");
if (y.Exists("a"))
document.write("true");
...
```

JScript Dictionary Object Add Method

Adds a key and item pair to a **Dictionary** object.

Syntax: JScript Dictionary Object Add Method

```
object.Add (key, item)
```

Arguments: JScript Dictionary Object Add Method

object

The name of a **Dictionary** object. Required.

key

The *key* associated with the *item* being added. Required.

item

The *item* associated with the *key* being added. Required.

Remarks: JScript Dictionary Object Add Method

An error occurs if the *key* already exists.

JScript Dictionary Object Count Property

Returns the number of items in a **Dictionary** object. Read-only.

Syntax: JScript Dictionary Object Count Property

```
object.Count
```

object

The name of a **Dictionary** object.

Remarks: JScript Dictionary Object Count Property

The following code illustrates use of the **Count** property:

```
var a, d, i, s; // Create some variables.
d = new ActiveXObject("Scripting.Dictionary");
d.Add ("a", "Athens"); // Add some keys and items
d.Add ("b", "Belgrade");
d.Add ("c", "Cairo");
a = (new VBArray(d.Keys())); // Get the keys.
s = "";
for (i = 0; i < d.Count; i++) //Iterate the dictionary.
{
s += a.getItem(i) + " - " + d(a.getItem(i)) + "<br>";
}
document.write(s); // Print item.
```

JScript Dictionary Object Exists Method

Returns **True** if a specified key exists in the **Dictionary** object, **False** if it does not.

Syntax: JScript Dictionary Object Exists Method

```
object.Exists(key)
```

Arguments: JScript Dictionary Object Exists Method

object

The name of a **Dictionary** object. Required.

key

The *key* value being searched for in the **Dictionary** object. Required.

JScript Dictionary Object Item Property

Sets or returns an *item* for a specified *key* in a **Dictionary** object Read/write.

Syntax: JScript Dictionary Object Item Property

```
object.Item(key)[ = newitem]
```

object

The name of a **Dictionary** object. Required.

key

Index associated with the *item* being retrieved or added. Required.

newitem

If provided, *newitem* is the new value associated with the specified *key*. Optional.

Remarks: JScript Dictionary Object Item Property

If *key* is not found when changing an *item*, a new *key* is created with the specified *newitem*. If *key* is not found when attempting to return an existing *item,* a new *key* is created and its corresponding *item* is left empty.

JScript Dictionary Object Items Method

Returns an array containing all the items in a **Dictionary** object.

Syntax: JScript Dictionary Object Items Method

```
object.Items( )
```

Arguments: JScript Dictionary Object Items Method

object

The name of a **Dictionary** object.

Remarks: JScript Dictionary Object Items Method

The following code illustrates use of the **Items** method:

```
var a, d, i, s; // Create some variables.
d = new ActiveXObject("Scripting.Dictionary");
d.Add ("a", "Athens"); // Add some keys and items
d.Add ("b", "Belgrade");
d.Add ("c", "Cairo");
a = (new VBArray(d.Items())).toArray(); // Get the items.
```

```
s = "";
for (i in a) //Iterate the dictionary.
{
s += a[i] + "<br>";
}
document.write(s); // Print item.
```

JScript Dictionary Object Key Property

Sets a *key* in a **Dictionary** object.

Syntax: JScript Dictionary Object Key Property

```
object.Key(key) = newkey
```

object

The name of a **Dictionary** object. Required.

key

The *key* value being changed. Required.

newkey

A new value that replaces the specified *key*. Required.

Remarks: JScript Dictionary Object Key Property

If *key* is not found when changing a *key*, a new *key* is created and its associated *item* is left empty.

JScript Dictionary Object Keys Method

Returns an array containing all existing keys in a **Dictionary** object.

Syntax: JScript Dictionary Object Keys Method

```
object.Keys( )
```

Arguments: JScript Dictionary Object Keys Method

object

The name of a **Dictionary** object.

Remarks: JScript Dictionary Object Keys Method

The following code illustrates use of the **Keys** method:

```
var a, d, i, s; // Create some variables.
d = new ActiveXObject("Scripting.Dictionary");
d.Add ("a", "Athens"); // Add some keys and items
```

```
d.Add ("b", "Belgrade");
d.Add ("c", "Cairo");
a = (new VBArray(d.Keys())).toArray(); // Get the keys.
s = "";
for (i in a) //Iterate the dictionary.
{
s += a[i] + " - " + d(a[i]) + "<br>";
}
document.write(s); // Print item.
```

JScript Dictionary Object Remove Method

Removes a key, item pair from a **Dictionary** object.

Syntax: JScript Dictionary Object Remove Method

```
object.Remove(key)
```

Arguments: JScript Dictionary Object Remove Method

object

The name of a **Dictionary** object. Required.

key

The *key* associated with the key, item pair you want to remove from the **Dictionary** object. Required.

Remarks: JScript Dictionary Object Remove Method

An error occurs if the specified key, item pair does not exist.

The following code illustrates use of the **Remove** method:

```
var a, d, i, s; // Create some variables.
d = new ActiveXObject("Scripting.Dictionary");
d.Add ("a", "Athens"); // Add some keys and items
d.Add ("b", "Belgrade");
d.Add ("c", "Cairo");
...
d.Remove("b"); // Remove second pair.
```

JScript Dictionary Object RemoveAll Method

The **RemoveAll** method removes all key, item pairs from a **Dictionary** object.

Applies To: JScript Dictionary Object RemoveAll Method

Dictionary

Syntax: JScript Dictionary Object RemoveAll Method

```
object.RemoveAll( )
```

Arguments: JScript Dictionary Object RemoveAll Method

object

The name of a **Dictionary** object.

Remarks: JScript Dictionary Object RemoveAll Method

The following code illustrates use of the **RemoveAll** method:

```
var a, d, i; // Create some variables.
d = new ActiveXObject("Scripting.Dictionary");
d.Add ("a", "Athens"); // Add some keys and items.
d.Add ("b", "Belgrade");
d.Add ("c", "Cairo");
...
d.RemoveAll( ); // Clear the dictionary.
```

JScript Drive Object

JScript Drive Object

The **Drive** object provides access to the properties of a particular disk drive or network share.

Properties: JScript Drive Object

JScript Drive Object AvailableSpace Property	The amount of space available to a user. *This property is not currently supported on UNIX.*
JScript Drive Object DriveLetter Property	The drive letter of a physical drive or network share. *This property is not currently supported on UNIX.*
JScript Drive Object DriveType Property	The type of a drive. *This property is not currently supported on UNIX.*
JScript Drive Object FileSystem Property	The type of file system in use on the drive. *This property is not currently supported on UNIX.*
JScript Drive Object FreeSpace Property	The amount of free space available to a user on a specified drive or network share. *This*

	property is currently not supported on UNIX.
JScript Drive Object IsReady Property	Boolean value indicating the status of a drive.
JScript Drive Object Path Property	The file system path to the drive.
JScript Drive Object RootFolder Property	A **Folder** object representing the root folder of a drive.
JScript Drive Object SerialNumber Property	The decimal serial number used to uniquely identify a disk volume. *This property is not currently supported on UNIX.*
JScript Drive Object ShareName Property	The network share name for the drive. *This property is not currently supported on UNIX.*
JScript Drive Object TotalSize Property	The total size, in bytes, of a drive or network share. *This property is not currently supported on UNIX.*
JScript Drive Object VolumeName Property	The volume name of a drive or network share. *This property is not currently supported on UNIX.*

Remarks: JScript Drive Object

The following code illustrates the use of the **Drive** object to access drive properties:

```
function ShowUsedSpace(drvPath)
{
var fs, d, s;
fs = new ActiveXObject("Scripting.FileSystemObject");
d = fs.GetDrive(fs.GetDriveName(drvPath));
s = "Space used on drive " + drvPath + " - " ;
s += d.RootFolder.Size/1024 + " Kbytes";
Response.Write(s);
}
```

JScript Drive Object AvailableSpace Property

Returns the amount of space available to a user on the specified drive or network share. *This property is not available under UNIX.*

Syntax: JScript Drive Object AvailableSpace Property

```
object.AvailableSpace
```

Arguments: JScript Drive Object AvailableSpace Property

object

A **Drive** object.

Remarks: JScript Drive Object AvailableSpace Property

The value returned by the **AvailableSpace** property is typically the same as that returned by the **FreeSpace** property. Differences may occur between the two for computer systems that support quotas.

The following code illustrates the use of the **AvailableSpace** property:

```
function ShowAvailableSpace(drvPath)
{
var fs, d, s;
fs = new ActiveXObject("Scripting.FileSystemObject");
d = fs.GetDrive(fs.GetDriveName(drvPath));
s = "Drive " + drvPath.toUpperCase + " - ";
s += d.VolumeName + "<br>";
s += "Available Space: " + d.AvailableSpace/1024 + " Kbytes";
Response.Write(s);
}
```

JScript Drive Object DriveLetter Property

Returns the drive letter of a physical local drive or a network share. *This property is not available under UNIX.* Read-only.

Syntax: JScript Drive Object DriveLetter Property

```
object.DriveLetter
```

Arguments: JScript Drive Object DriveLetter Property

object

A **Drive** object.

Remarks: JScript Drive Object DriveLetter Property

The **DriveLetter** property returns a zero-length string ("") if the specified drive is not associated with a drive letter, for example, a network share that has not been mapped to a drive letter.

The following code illustrates the use of the **DriveLetter** property:

```
function ShowDriveLetter(drvPath)
{
var fs, d, s;
fs = new ActiveXObject("Scripting.FileSystemObject");
```

```
d = fs.GetDrive(fs.GetDriveName(drvPath));
s = "Drive " + d.DriveLetter.toUpperCase( ) + ": - ";
s += d.VolumeName + "<br>";
s += "Available Space: " + d.AvailableSpace/1024 + " Kbytes";
Response.Write(s);
}
```

JScript Drive Object DriveType Property

Returns a value indicating the type of a specified drive. *This property is not available under UNIX.*

Syntax: JScript Drive Object DriveType Property

```
object.DriveType
```

Arguments: JScript Drive Object DriveType Property

object

A **Drive** object.

Remarks: JScript Drive Object DriveType Property

The following code illustrates the use of the **DriveType** property:

```
function ShowDriveType(drvpath)
{
var fs, d, s, t;
fs = new ActiveXObject("Scripting.FileSystemObject");
d = fs.GetDrive(drvpath);
switch (d.DriveType)
{
case 0: t = "Unknown"; break;
case 1: t = "Removable"; break;
case 2: t = "Fixed"; break;
case 3: t = "Network"; break;
case 4: t = "CD-ROM"; break;
case 5: t = "RAM Disk"; break;
}
s = "Drive " + d.DriveLetter + ": - " + t;
```

```
Response.Write(s);
}
```

JScript Drive Object FileSystem Property

Returns the type of file system in use for the specified drive. *This property is not available under UNIX.*

Syntax: JScript Drive Object FileSystem Property

```
object.FileSystem
```

Arguments: JScript Drive Object FileSystem Property

object

A **Drive** object.

Remarks: JScript Drive Object FileSystem Property

Available return types include FAT, NTFS, and CDFS.

The following code illustrates the use of the **FileSystem** property:

```
function ShowFileSystemType(drvPath)
{
var fs,d, s;
fs = new ActiveXObject("Scripting.FileSystemObject");
d = fs.GetDrive(drvPath);
s = d.FileSystem;
Response.Write(s);
}
```

JScript Drive Object FreeSpace Property

Returns the amount of free space available to a user on the specified drive or network share. *This property is not available under UNIX.* Read-only.

Syntax: JScript Drive Object FreeSpace Property

```
object.FreeSpace
```

Arguments: JScript Drive Object FreeSpace Property

object

A **Drive** object.

Remarks: JScript Drive Object FreeSpace Property

The value returned by the **FreeSpace** property is typically the same as that returned by the **AvailableSpace** property. Differences may occur between the two for computer systems that support quotas.

The following code illustrates the use of the **FreeSpace** property:

```
function ShowFreeSpace(drvPath)
{
var fs, d, s;
fs = new ActiveXObject("Scripting.FileSystemObject");
d = fs.GetDrive(fs.GetDriveName(drvPath));
s = "Drive " + drvPath.toUpperCase( ) + " - ";
s += d.VolumeName + "<br>";
s += "Free Space: " + d.FreeSpace/1024 + " Kbytes";
Response.Write(s);
}
```

JScript Drive Object IsReady Property

Returns **True** if the specified drive is ready; **False** if it is not.

Syntax: JScript Drive Object IsReady Property

```
object.IsReady
```

Arguments: JScript Drive Object IsReady Property

object

A **Drive** object.

Remarks: JScript Drive Object IsReady Property

For removable-media drives and CD-ROM drives on Windows systems, **IsReady** returns **True** only when the appropriate media is inserted and ready for access. Under UNIX, **IsReady** always returns **True.**

The following code illustrates the use of the IsReady property:

```
function ShowDriveInfo(drvpath)
{
var fs, d, s, t;
fs = new ActiveXObject("Scripting.FileSystemObject")
d = fs.GetDrive(drvpath)
switch (d.DriveType)
```

```
{
case 0: t = "Unknown"; break;
case 1: t = "Removable"; break;
case 2: t = "Fixed"; break;
case 3: t = "Network"; break;
case 4: t = "CD-ROM"; break;
case 5: t = "RAM Disk"; break;
}
s = "Drive " + d.DriveLetter + ": - " + t;
if (d.IsReady)
{
s += "<br>" + "Drive is Ready.";
}
else
{
s += "<br>" + "Drive is not Ready.";
}
Response.Write(s);
}
```

JScript Drive Object Path Property

Returns the path for a specified drive.

Syntax: JScript Drive Object Path Property

```
object.Path
```

Arguments: JScript Drive Object Path Property

object

A **Drive** object.

Remarks: JScript Drive Object Path Property

For drive letters, the root drive is not included. For example, the path for the C drive is C:, not C:\.

JScript Drive Object RootFolder Property

Returns a **Folder** object representing the root folder of a specified drive. Read-only.

Syntax: JScript Drive Object RootFolder Property

object.**RootFolder**

Arguments: JScript Drive Object RootFolder Property

object

A **Drive** object.

Remarks: JScript Drive Object RootFolder Property

All the files and folders contained on the drive can be accessed using the returned **Folder** object.

JScript Drive Object SerialNumber Property

Returns the decimal serial number used to uniquely identify a disk volume. *This property is not available under UNIX.*

Syntax: JScript Drive Object SerialNumber Property

object.**SerialNumber**

Arguments: JScript Drive Object SerialNumber Property

object

A **Drive** object.

Remarks: JScript Drive Object SerialNumber Property

You can use the **SerialNumber** property to ensure that the correct disk is inserted in a drive with removable media.

The following code illustrates the use of the **SerialNumber** property:

```
function ShowDriveInfo(drvpath)
{
var fs, d, s, t;
fs = new ActiveXObject("Scripting.FileSystemObject");
d = fs.GetDrive(fs.GetDriveName(fs.GetAbsolutepathname(drvpath)));
switch (d.DriveType)
{
case 0: t = "Unknown"; break;
case 1: t = "Removable"; break;
case 2: t = "Fixed"; break;
case 3: t = "Network"; break;
```

```
case 4: t = "CD-ROM"; break;
case 5: t = "RAM Disk"; break;
}
s = "Drive " + d.DriveLetter + ": - " + t;
s += "<br>" + "SN: " + d.SerialNumber;
Response.Write(s);
}
```

JScript Drive Object ShareName Property

Returns the network share name for a specified drive. *This property is not available under UNIX.*

Syntax: JScript Drive Object ShareName Property

```
object.ShareName
```

Arguments: JScript Drive Object ShareName Property

object

A **Drive** object.

Remarks: JScript Drive Object ShareName Property

If *object* is not a network drive, the **ShareName** property returns a zero-length string ("").

The following code illustrates the use of the **ShareName** property:

```
function ShowDriveInfo(drvpath)
{
var fs, d, s;
fs = new ActiveXObject("Scripting.FileSystemObject");
d = fs.GetDrive(fs.GetDriveName(fs.GetAbsolutepathname(drvpath)));
s = "Drive " + d.DriveLetter + ": - " + d.ShareName;
Response.Write(s);
}
```

JScript Drive Object TotalSize Property

Returns the total space, in bytes, of a drive or network share. *This property is not available under UNIX.*

Syntax: JScript Drive Object TotalSize Property

```
object.TotalSize
```

Arguments: JScript Drive Object TotalSize Property

object

A **Drive** object.

Remarks: JScript Drive Object TotalSize Property

The following code illustrates the use of the **TotalSize** property:

```
function SpaceReport(drvPath)
{
var fs, d, s;
fs = new ActiveXObject("Scripting.FileSystemObject");
d = fs.GetDrive(fs.GetDriveName(drvPath));
s = "Drive " + drvPath + " - ";
s += d.VolumeName + "<br>";
s += "Total Space: "+ d.TotalSize/1024 + " Kbytes <br>";
s += "Free Space: " + d.FreeSpace/1024 + " Kbytes";
Response.Write(s);
}
```

JScript Drive Object VolumeName Property

Sets or returns the volume name of the specified drive. *This property is not available under UNIX.* Read/write.

Syntax: JScript Drive Object VolumeName Property

```
object.VolumeName [= newname]
```

Arguments: JScript Drive Object VolumeName Property

object

Required. Always the name of a **Drive** object.

newname

The new name of the specified object. Optional.

Remarks: JScript Drive Object VolumeName Property

The following code illustrates the use of the **VolumeName** property:

```
function SpaceReport(drvPath)
{
var fs, d, s;
```

```
fs = new ActiveXObject("Scripting.FileSystemObject");
d = fs.GetDrive(fs.GetDriveName(drvPath));
s = "Drive " + drvPath + " - ";
s += d.VolumeName + "<br>";
s += "Total Space: "+ d.TotalSize/1024 + " Kbytes <br>";
s += "Free Space: " + d.FreeSpace/1024 + " Kbytes";
Response.Write(s);
}
```

JScript Enumerator Object

JScript Enumerator Object

The **Enumerator** object provides a way to enumerate items in a collection.

Methods: JScript Enumerator Object

JScript Enumerator Object AtEnd Method	Returns a Boolean value indicating if an **Enumerator** object is at the end of a collection.
JScript Enumerator Object item Method	Returns the current item in the collection.
JScript Enumerator Object moveFirst Method	Resets the current item pointer to the first item in the collection.
JScript Enumerator Object moveNext Method	Moves the current item pointer to the next item in the collection.

Syntax: JScript Enumerator Object

```
new Enumerator(collection)
```

Arguments: JScript Enumerator Object

collection

Any collection object.

Remarks: JScript Enumerator Object

Collections differ from arrays in that the members of a collection are not directly accessible. Instead of using indices, as you would with arrays, you can only move the current item pointer to the first or next element of a collection.

The **Enumerator** object provides a way to access any member of a collection and behaves similarly to the **for…in** statement in JScript.

The following code shows the usage of the **Enumerator** object:

```
function ShowDriveList()
{
var fs, s, n, e, x;
fs = new ActiveXObject("Scripting.FileSystemObject");
e = new Enumerator(fs.Drives);
s = "";
for (;!e.atEnd();e.moveNext())
{
x = e.item();
s = s + x.DriveLetter;
s += " - ";
if (x.DriveType == 3)
n = x.ShareName;
else if (x.IsReady)
n = x.VolumeName;
else
n = "[Drive not ready]";
s += n + "<br>";
}
Response.Write(s);
}
```

JScript Enumerator Object AtEnd Method

Returns a Boolean value indicating if the enumerator is at the end of the collection.

Syntax: JScript Enumerator Object AtEnd Method

myEnum.**atEnd( )**

Arguments: JScript Enumerator Object AtEnd Method

myEnum

Any **Enumerator** object.

Return Value: JScript Enumerator Object AtEnd Method

The **atEnd** method returns **True** if the current item is the last one in the collection, the collection is empty, or the current item is undefined. Otherwise, it returns **False**.

JScript Enumerator Object item Method

Returns the current item in the collection.

Syntax: JScript Enumerator Object item Method

```
myEnum.item()
```

Parameters[0]: JScript Enumerator Object item Method

myEnum

Any **Enumerator** object.

Return Value: JScript Enumerator Object item Method

The **item** method returns the current item. If the collection is empty or if the current item is undefined, **item** returns undefined.

JScript Enumerator Object moveFirst Method

Resets the current item in the collection to the first item.

Syntax: JScript Enumerator Object moveFirst Method

```
myEnum.moveFirst( )
```

Arguments: JScript Enumerator Object moveFirst Method

myEnum

Any **Enumerator** object.

Remarks: JScript Enumerator Object moveFirst Method

If there are no items in the collection, the current item is set to undefined.

JScript Enumerator Object moveNext Method

Moves the current item to the next item in the collection.

Syntax: JScript Enumerator Object moveNext Method

```
myEnum.moveNext( )
```

Arguments: JScript Enumerator Object moveNext Method

myEnum

Any **Enumerator** object.

Remarks: JScript Enumerator Object moveNext Method

If the enumerator is at the end of the collection or the collection is empty, the current item is set to undefined.

JScript File Object

JScript File Object

The **File** object provides access to all the properties of a file.

Methods: JScript File Object

JScript File Object OpenAsTextStream Method	Opens a file and returns a **TextStream** object.
JScript File Object Copy Method	Copies a file from one location to another.
JScript File Object Delete Method	Deletes a file.
JScript File Object Move Method	Moves a file from one location to another.

Properties: JScript File Object

JScript File Object Attributes Property	The file system attributes of a file.
JScript File Object DateCreated Property	The date and time that a file was created.
JScript File Object DateLastAccessed Property	The date and time that a file was last accessed.
JScript File Object DateLastModified Property	The date and time that a file was last modified.
JScript File Object Drive Property	The drive letter of the drive on which the file resides. *On UNIX, this property is always '/'.*
JScript File Object Name Property	The name of a file.
JScript File Object ParentFolder Property	The **Folder** object containing the file.
JScript File Object Path Property	The file system path for the file.
JScript File Object ShortName Property	The short name used by programs that require 8.3 file names. *This property is not currently supported on UNIX.*
JScript File Object ShortPath Property	The short path used by programs that require 8.3 file names. *This property is not currently supported on UNIX.*
JScript File Object Size Property	The size, in bytes, of a file.
JScript File Object Type Property	Information about the type of a file. *This property is not currently supported on UNIX.*

Remarks: JScript File Object

The following code illustrates how to obtain a **File** object and how to view one of its properties.

```
function ShowFileInfo(filespec)
{
```

```
var fs, file, s;
fs = new ActiveXObject("Scripting.FileSystemObject");
file = fs.GetFile(filespec);
s = file.DateCreated;
Response.Write(s);
}
```

JScript File Object Attributes Property

Sets or returns the file system attributes of files. Read/write or read-only, depending on the attribute.

Note

This property depends on the underlying operating system for its behavior. If the OS file system does not support the file attribute requested, an error will be returned.

Syntax: JScript File Object Attributes Property

```
object.Attributes [= newattributes]
```

Arguments: JScript File Object Attributes Property

object

The name of a **File** object. Required.

newattributes

If provided, *newattributes* is the new value for the attributes of the specified *object*. Optional.

Settings: JScript File Object Attributes Property

The *newattributes* argument can have any of the following values or any logical combination of the following values:

Constant	Value	Description
Normal	0	Normal file. No attributes are set.
ReadOnly	1	Read-only file. Attribute is read/write.
Hidden	2	Hidden file. Attribute is read/write.
System	4	System file. Attribute is read/write.
Volume	8	Disk drive volume label. Attribute is read-only.
Directory	16	Folder or directory. Attribute is read-only.
Archive	32	File has changed since last backup. Attribute is read/write.

Alias	64	Link or shortcut. Attribute is read-only.
Compressed	128	Compressed file. Attribute is read-only.

Remarks: JScript File Object Attributes Property

The following code illustrates the use of the **Attributes** property with a file:

```
function ToggleArchiveBit(filespec)
{
var fs, f, r, s;
fs = new ActiveXObject("Scripting.FileSystemObject");
f = fs.GetFile(filespec)
if (f.attributes && 32)
{
f.attributes = f.attributes - 32;
s = "Archive bit is cleared.";
}
else
{
f.attributes = f.attributes + 32;
s = "Archive bit is set.";
}
return s;
}
```

JScript File Object Copy Method

Copies a specified file from one location to another.

Syntax: JScript File Object Copy Method

```
object.Copy( destination[, overwrite] );
```

Arguments: JScript File Object Copy Method

object

The name of a **File** object. Required.

destination

The destination where the file is to be copied. Wildcard characters are not allowed. Required.

overwrite

A **Boolean** value that is **True** (default) if existing files are to be overwritten; **False** if they are not. Optional.

Remarks: JScript File Object Copy Method

The results of the **Copy** method on a **File** are identical to operations performed using **CopyFile** where the file referred to by *object* is passed as an argument. You should note, however, that the alternative method is capable of copying multiple files.

JScript File Object DateCreated Property

Returns the date and time that the specified file was created. Read-only.

Syntax: JScript File Object DateCreated Property

```
object.DateCreated
```

Arguments: JScript File Object DateCreated Property

object

A **File** object.

Remarks: JScript File Object DateCreated Property

The following code illustrates the use of the **DateCreated** property with a file:

```
function ShowFileInfo(filespec)
{
var fs, f, s;
fs = new ActiveXObject("Scripting.FileSystemObject");
f = fs.GetFile(filespec);
s = "Created: " + f.DateCreated;
Response.Write(s);
}
```

JScript File Object DateLastAccessed Property

Returns the date and time that the specified file was last accessed. Read-only.

Syntax: JScript File Object DateLastAccessed Property

```
object.DateLastAccessed
```

Arguments: JScript File Object DateLastAccessed Property

object

A **File** object.

Remarks: JScript File Object DateLastAccessed Property

The following code illustrates the use of the **DateLastAccessed** property with a file:

```
function ShowFileAccessInfo(filespec)
{
var fs, f, s;
fs = new ActiveXObject("Scripting.FileSystemObject");
f = fs.GetFile(filespec);
s = filespec.toUpperCase() + "<br>";
s += "Created: " + f.DateCreated + "<br>";
s += "Last Accessed: " + f.DateLastAccessed + "<br>";
s += "Last Modified: " + f.DateLastModified;
Response.Write(s);
}
```

Note

This method depends on the underlying operating system for its behavior. If the operating system does not support providing time information, none will be returned.

JScript File Object DateLastModified Property

Returns the date and time that the specified file was last modified. Read-only.

Syntax: JScript File Object DateLastModified Property

```
object.DateLastModified
```

Arguments: JScript File Object DateLastModified Property

object

A **File** object.

Remarks: JScript File Object DateLastModified Property

The following code illustrates the use of the **DateLastModified** property with a file:

```
function ShowFileAccessInfo(filespec)
{
var fs, f, s;
fs = new ActiveXObject("Scripting.FileSystemObject");
f = fs.GetFile(filespec);
s = filespec.toUpperCase() + "<br>";
```

```
s += "Created: " + f.DateCreated + "<br>";
s += "Last Accessed: " + f.DateLastAccessed + "<br>";
s += "Last Modified: " + f.DateLastModified;
Response.Write(s);
}
```

JScript File Object Delete Method

Deletes a specified file or folder.

Syntax: JScript File Object Delete Method

```
object.Delete( force );
```

Arguments: JScript File Object Delete Method

object

Required. Always the name of a **File** object.

force

Optional. Boolean value that is **True** if files with the read-only attribute set are to be deleted, **False** (default) if they are not.

Remarks: JScript File Object Delete Method

An error occurs if the specified file does not exist.

The results of the **Delete** method on a **File** are identical to operations performed using **DeleteFile**.

JScript File Object Drive Property

Returns the drive letter of the drive on which the specified file resides. Read-only.

Syntax: JScript File Object Drive Property

```
object.Drive
```

Arguments: JScript File Object Drive Property

object

A **File** object.

Remarks: JScript File Object Drive Property

On UNIX systems, the **Drive** property is always "/".

The following code illustrates the use of the **Drive** property on Windows systems:

```
function ShowFileAccessInfo(filespec)
{
```

```
var fs, f, s;
fs = new ActiveXObject("Scripting.FileSystemObject");
f = fs.GetFile(filespec);
s = f.Name + " on Drive " + f.Drive + "<br>";
s += "Created: " + f.DateCreated + "<br>";
s += "Last Accessed: " + f.DateLastAccessed + "<br>";
s += "Last Modified: " + f.DateLastModified;
Response.Write(s);
}
```

JScript File Object Move Method

Moves a specified file from one location to another.

Syntax: JScript File Object Move Method

```
object.Move( destination );
```

Arguments: JScript File Object Move Method

object

The name of a **File** object. Required.

destination

Destination where the file is to be moved. Wildcard characters are not allowed. Required.

Remarks: JScript File Object Move Method

The results of the **Move** method on a **File** are identical to operations performed using **MoveFile**. You should note, however, that the alternative method is capable of moving multiple files.

JScript File Object Name Property

Sets or returns the name of a specified file. Read/write.

Syntax: JScript File Object Name Property

```
object.Name [= newname]
```

Arguments: JScript File Object Name Property

object

The name of a **File** object. Required.

newname

The new name of the specified *object*, if provided. Optional.

Remarks: JScript File Object Name Property

The following code illustrates the use of the **Name** property:

```
function ShowFileAccessInfo(filespec)
{
var fs, f, s;
fs = new ActiveXObject("Scripting.FileSystemObject");
f = fs.GetFile(filespec);
s = f.Name + " on Drive " + f.Drive + "<br>";
s += "Created: " + f.DateCreated + "<br>";
s += "Last Accessed: " + f.DateLastAccessed + "<br>";
s += "Last Modified: " + f.DateLastModified;
Response.Write(s);
}
```

JScript File Object OpenAsTextStream Method

Opens a specified file and returns a **TextStream** object that can be used to read from, write to, or append to the file.

Syntax: JScript File Object OpenAsTextStream Method

```
object.OpenAsTextStream([iomode, [format]])
```

Arguments: JScript File Object OpenAsTextStream Method

object

The name of a **File** object. Required.

iomode

Indicates input/output mode. Can be one of three constants: **ForReading**, **ForWriting**, or **ForAppending**. Optional.

format

One of three **Tristate** values used to indicate the format of the opened file. If omitted, the file is opened as ASCII. Optional.

Settings: JScript File Object OpenAsTextStream Method

The *iomode* argument can have any of the following settings:

Constant	Value	Description
ForReading	1	Open a file for reading only. You can't write to this file.

ForWriting	2	Open a file for writing. If a file with the same name exists, its previous contents are overwritten.
ForAppending	8	Open a file and write to the end of the file.

The *format* argument can have any of the following settings:

Constant	Value	Description
TristateUseDefault	-2	Opens the file using the system default.
TristateTrue	-1	Opens the file as Unicode.
TristateFalse	0	Opens the file as ASCII.

Remarks: JScript File Object OpenAsTextStream Method

The **OpenAsTextStream** method provides the same functionality as the **OpenTextFile** method of the **FileSystemObject**. In addition, the **OpenAsTextStream** method can be used to write to a file.

The following code illustrates the use of the **OpenAsTextStream** method:

```
function TextStreamTest( )
{
var ForReading = 1, ForWriting = 2, ForAppending = 3;
var TristateUseDefault = -2, TristateTrue = -1, TristateFalse = 0;
var fs, f, ts, s;
fs = new ActiveXObject("Scripting.FileSystemObject");
fs.CreateTextFile( "test1.txt" ); // Create a file
f = fs.GetFile("test1.txt");
ts = f.OpenAsTextStream(ForWriting, TristateUseDefault);
ts.Write( "Hello World" );
ts.Close( );
ts = f.OpenAsTextStream(ForReading, TristateUseDefault);
s = ts.ReadLine( );
ts.Close( );
Response.Write(s);
}
```

JScript File Object ParentFolder Property

Returns the folder object for the parent of the specified file. Read-only.

Syntax: JScript File Object ParentFolder Property

```
object.ParentFolder
```

Arguments: JScript File Object ParentFolder Property

object

A **File** object.

Remarks: JScript File Object ParentFolder Property

The following code illustrates the use of the **ParentFolder** property with a file:

```
function ShowFileAccessInfo(filespec)
{
var fs, f, s;
fs = new ActiveXObject("Scripting.FileSystemObject");
f = fs.GetFile(filespec);
s = f.Name + " in " + f.ParentFolder + "<br>";
s += "Created: " + f.DateCreated + "<br>";
s += "Last Accessed: " + f.DateLastAccessed + "<br>";
s += "Last Modified: " + f.DateLastModified;
Response.Write(s);
}
```

JScript File Object Path Property

Returns the path for a specified file.

Syntax: JScript File Object Path Property

```
object.Path
```

Arguments: JScript File Object Path Property

object

A **File** object.

Remarks: JScript File Object Path Property

The following code illustrates the use of the **Path** property with a **File** object:

```
function ShowFileAccessInfo(filespec)
{
var fs, d, f, s;
fs = new ActiveXObject("Scripting.FileSystemObject");
```

```
f = fs.GetFile(filespec);
s = f.Path.toUpperCase() + "<br>";
s += "Created: " + f.DateCreated + "<br>";
s += "Last Accessed: " + f.DateLastAccessed + "<br>";
s += "Last Modified: " + f.DateLastModified
Response.Write(s);
}
```

JScript File Object ShortName Property

Returns the short name used by programs that require the earlier 8.3 naming convention. *This property is not available under UNIX.*

Syntax: JScript File Object ShortName Property

```
object.ShortName
```

Arguments: JScript File Object ShortName Property

object

A **File** object.

Remarks: JScript File Object ShortName Property

The following code illustrates the use of the **ShortName** property with a **File** object:

```
function ShowShortName(filespec)
{
var fs, f, s;
fs = new ActiveXObject("Scripting.FileSystemObject");
f = fs.GetFile(filespec);
s = "The short name for " + "" + f.Name;
s += "" + "<br>";
s += "is: " + "" + f.ShortName + "";
Response.Write(s);
}
```

JScript File Object ShortPath Property

Returns the short path used by programs that require the earlier 8.3 file naming convention. *This property is not available under UNIX.*

Syntax: JScript File Object ShortPath Property

object.**ShortPath**

object

A **File** object.

Remarks: JScript File Object ShortPath Property

The following code illustrates the use of the **ShortPath** property with a **File** object:

```
function ShowShortPath(filespec);
{
var fs, f, s;
fs = new ActiveXObject("Scripting.FileSystemObject");
f = fs.GetFile(filespec);
s = "The short path for " + "" + f.Name;
s += "" + "<br>";
s += "is: " + "" + f.ShortPath + "";
Response.Write(s);
}
```

JScript File Object Size Property

The size, in bytes, of the specified file

Syntax: JScript File Object Size Property

object.**Size**

Arguments: JScript File Object Size Property

object

A **File** object.

Remarks: JScript File Object Size Property

The following code illustrates the use of the **Size** property with a **File** object:

```
function ShowFolderSize(filespec)
{
var fs, f, s;
fs = new ActiveXObject("Scripting.FileSystemObject");
f = fs.GetFile(filespec);
s = f.Name + " uses " + f.size + " bytes.";
```

```
Response.Write(s);
}
```

JScript File Object Type Property

Returns information about the type of a file. For example, for files ending in .TXT, "Text Document" is returned. *This property is not available under UNIX.*

Syntax: JScript File Object Type Property

```
object.Type
```

Arguments: JScript File Object Type Property

object

A **File** object.

Remarks: JScript File Object Type Property

The following code illustrates the use of the **Type** property to return a file type.

```
function ShowFileType(filespec)
{
var fs, f, s;
fs = new ActiveXObject("Scripting.FileSystemObject");
if (fs.FolderExists(filespec))
f = fs.GetFolder(filespec);
else if (fs.FileExists(filespec))
f = fs.GetFile(filespec);
else
s = "File or Folder does not exist.";
s = f.Name + " is a " + f.Type;
Response.Write(s);
}
```

JScript FileSystemObject Object

JScript FileSystemObject Object

The **FileSystemObject** provides access to a computer's file system.

Methods: JScript FileSystemObject Object

JScript FileSystemObject Object BuildPath Method	Appends a name to an existing path.
JScript FileSystemObject Object CopyFile Method	Copies one or more files from one location to another.
JScript FileSystemObject Object CopyFolder Method	Copies one or more folders and their contents from one location to another.
JScript FileSystemObject Object CreateFolder Method	Creates a folder
JScript FileSystemObject Object CreateTextFile Method	Creates a text file with a specified name and returns a **TextStream** object.
JScript FileSystemObject Object DeleteFile Method	Deletes a specified file.
JScript FileSystemObject Object DeleteFolder Method	Deletes a specified folder and its contents.
JScript FileSystemObject Object DriveExists Method	Returns a Boolean value indicating the existence of a drive.
JScript FileSystemObject Object FileExists Method	Returns a Boolean value indicating the existence of a file.
JScript FileSystemObject Object FolderExists Method	Returns a Boolean value indicating the existence of a folder.
JScript FileSystemObject Object GetAbsolutepathname Method	Returns a complete and unambiguous path from a provided path specification.
JScript FileSystemObject Object GetBaseName Method	Returns a string containing the base name of the last component, less any extension, in a path.
JScript FileSystemObject Object GetDrive Method	Returns a **Drive** object corresponding to the drive in a specified path.
JScript FileSystemObject Object GetDriveName Method	Returns a string containing the name of a drive for a specified path.
JScript FileSystemObject Object GetExtensionName Method	Returns a string containing the extension name for the last component in a path.
JScript FileSystemObject Object GetFile Method	Returns a **File** object corresponding to the file in a specified path.
JScript FileSystemObject Object GetFileName Method	Returns the last component of the specified path that is not part of the drive specification.
JScript FileSystemObject Object GetFolder Method	Returns a **Folder** object corresponding to the folder in a specified path.
JScript FileSystemObject Object	Returns a string containing the name of the

GetParentFolderName Method	parent folder of the last component in a specified path.
JScript FileSystemObject Object GetSpecialFolder Method	Returns the special folder specified.
JScript FileSystemObject Object GetTempName Method	Returns a randomly generated temporary file or folder name.
JScript FileSystemObject Object MoveFile Method	Moves one or more files from one location to another.
JScript FileSystemObject Object MoveFolder Method	Moves one or more folders and their contents from one location to another.
JScript FileSystemObject Object OpenTextFile Method	Opens a specified file and returns a **TextStream** object.

Properties: JScript FileSystemObject Object

JScript FileSystemObject Object Drives Property	A collection of all **Drive** objects available on the local machine. See the **Drives** collection section.

Note

Collections returned by **FileSystemObject** method calls reflect the state of the file system when the collection was created. Changes to the file system after creation are not reflected in the collection. If the file system might be changed during the lifetime of the collection object, the method returning the collection should be called again to ensure that the contents are current.

Remarks: JScript FileSystemObject Object

The following code illustrates how the **FileSystemObject** is used to return a **TextStream** object that can be read from or written to:

```
var fs = new ActiveXObject("Scripting.FileSystemObject");
var a = fs.CreateTextFile("c:\testfile.txt", true);
a.WriteLine("This is a test.");
a.Close();
```

JScript FileSystemObject Object BuildPath Method

Appends a name to an existing path.

Syntax: JScript FileSystemObject Object BuildPath Method

```
object.BuildPath(path, name)
```

Arguments: JScript FileSystemObject Object BuildPath Method

object

Always the name of a **FileSystemObject** object. Required.

path

Existing path to which *name* is appended. Path can be absolute or relative and need not specify an existing folder. Required.

name

Name being appended to the existing *path*. Required.

Remarks: JScript FileSystemObject Object BuildPath Method

The **BuildPath** method inserts an additional path separator between the existing *path* and the *name,* only if necessary.

JScript FileSystemObject Object CopyFile Method

Copies one or more files from one location to another.

Syntax: JScript FileSystemObject Object CopyFile Method

```
object.CopyFile ( source, destination[, overwrite] )
```

Arguments: JScript FileSystemObject Object CopyFile Method

object

The name of a **FileSystemObject** object. Required.

source

A character string file specification, which can include wildcard characters, for one or more files to be copied. Required.

destination

A character string destination where the file or files from *source* are to be copied. Wildcard characters are not allowed. Required.

overwrite

A **Boolean** value that indicates if existing files are to be overwritten. If true, files are overwritten; if false, they are not. The default is true.

Note

CopyFile **will fail if *destination* has the read-only attribute set, regardless of the value of *overwrite*. Optional**.

Remarks: JScript FileSystemObject Object CopyFile Method

Wildcard characters can only be used in the last path component of the *source* argument. For example, you can use:

```
fs = new ActiveXObject("Scripting.FileSystemObject");
fs.CopyFile ("c:\\mydocuments\\letters\\*.doc",
  "c:\\tempfolder\\")
But you can't use:
fs = new ActiveXObject("Scripting.FileSystemObject");
fs.CopyFile ("c:\\mydocuments\\*\\R1???97.xls",
  "c:\\tempfolder")
```

If *source* contains wildcard characters or *destination* ends with a path separator (\), it is assumed that *destination* is an existing folder in which to copy matching files. Otherwise, *destination* is assumed to be the name of a file to create. In either case, three things can happen when an individual file is copied:

- If destination does not exist, *source* gets copied. This is the usual case.
- If destination is an existing file, an error occurs if *overwrite* is false. Otherwise, an attempt is made to copy source over the existing file.
- If *destination* is a directory, an error occurs.

An error also occurs if a *source* using wildcard characters doesn't match any files. The **CopyFile** method stops on the first error it encounters. No attempt is made to roll back or undo any changes made before an error occurs.

JScript FileSystemObject Object CopyFolder Method

Recursively copies a folder from one location to another.

Syntax: JScript FileSystemObject Object CopyFolder Method

```
object.CopyFolder ( source, destination[, overwrite] );
```

Arguments: JScript FileSystemObject Object CopyFolder Method

object

The name of a **FileSystemObject**. Required.

source

A character string folder specification, which can include wildcard characters, for one or more folders to be copied. Required.

destination

A character string destination where the folder and subfolders from *source* are to be copied. Wildcard characters are not allowed. Required.

overwrite

A **Boolean** value that indicates if existing folders are to be overwritten. If **True**, files are overwritten; if **False**, they are not. The default is **True**. Optional.

Remarks: JScript FileSystemObject Object CopyFolder Method

Wildcard characters can only be used in the last path component of the *source* argument. For example, you can use:

```
fs = new ActiveXObject("Scripting.FileSystemObject");
fs.CopyFolder ("c:\\mydocuments\\letters\\*",
"c:\\tempfolder\\")
But you can't use:
fs = new ActiveXObject("Scripting.FileSystemObject");
fs.CopyFolder ("c:\\mydocuments\\*\\*", "c:\\tempfolder\\")
```

If *source* contains wildcard characters or *destination* ends with a path separator (\), it is assumed that *destination* is an existing folder in which to copy matching folders and subfolders. Otherwise, *destination* is assumed to be the name of a folder to create. In either case, four things can happen when an individual folder is copied:

- If *destination* does not exist, the source folder and all its contents gets copied. This is the usual case.
- If *destination* is an existing file, an error occurs.
- If *destination* is a directory, an attempt is made to copy the folder and all its contents. If a file contained in *source* already exists in *destination*, an error occurs if *overwrite* is **false**. Otherwise, it will attempt to copy the file over the existing file.
- If *destination* is a read-only directory, an error occurs if an attempt is made to copy an existing read-only file into that directory and *overwrite* is **false**.

An error also occurs if a source using wildcard characters doesn't match any folders.

The **CopyFolder** method stops on the first error it encounters. No attempt is made to roll back any changes made before an error occurs.

JScript FileSystemObject Object CreateFolder Method

Creates a folder.

Syntax: JScript FileSystemObject Object CreateFolder Method

```
object.CreateFolder(foldername)
```

Arguments: JScript FileSystemObject Object CreateFolder Method

object

The name of a **FileSystemObject** object. Required.

foldername

Required. **String** expression that identifies the folder to create.

Remarks: JScript FileSystemObject Object CreateFolder Method

An error occurs if the specified folder already exists.

The following code illustrates how to use the **CreateFolder** method to create a folder:

```
var fs = new ActiveXObject("Scripting.FileSystemObject");
var a = fs.CreateFolder("c:\\new folder");
```

JScript FileSystemObject Object CreateTextFile Method

Creates a specified file name and returns a **TextStream** object that can be used to read from or write to the file.

Syntax: JScript FileSystemObject Object CreateTextFile Method

```
object.CreateTextFile(filename[, overwrite[, unicode]])
```

Arguments: JScript FileSystemObject Object CreateTextFile Method

object

The name of a **FileSystemObject** object. Required.

filename

A **String** expression that identifies the file to create. Required.

overwrite

A **Boolean** value that indicates whether you can overwrite an existing file. The value is **true** if the file can be overwritten, **false** if it can't be overwritten. If omitted, existing files are not overwritten. Optional.

unicode

A **Boolean** value that indicates whether the file is created as a Unicode or ASCII file. The value is **true** if the file is created as a Unicode file, **false** if it's created as an ASCII file. If omitted, an ASCII file is assumed. Optional.

Remarks: JScript FileSystemObject Object CreateTextFile Method

The following code illustrates how to use the **CreateTextFile** method to create and open a text file:

```
var fs = new ActiveXObject("Scripting.FileSystemObject");
var a = fs.CreateTextFile("c:\\testfile.txt", true);
a.WriteLine("This is a test.");
a.Close();
```

If the *overwrite* argument is **False**, or is not provided, for a *filename* that already exists, an error occurs.

JScript FileSystemObject Object DeleteFile Method

Deletes a specified file.

Syntax: JScript FileSystemObject Object DeleteFile Method

```
object.DeleteFile ( filespec[, force] );
```

Arguments: JScript FileSystemObject Object DeleteFile Method

object

The name of a **FileSystemObject** object. Required.

filespec

The name of the file to delete. The *filespec* can contain wildcard characters in the last path component. Required.

force

A **Boolean** value that is **true** if files with the read-only attribute set are to be deleted; **false** (default) if they are not. Optional.

Remarks: JScript FileSystemObject Object DeleteFile Method

An error occurs if no matching files are found. The **DeleteFile** method stops on the first error it encounters. No attempt is made to roll back or undo any changes that were made before an error occurred.

JScript FileSystemObject Object DeleteFolder Method

Deletes a specified folder and its contents.

Syntax: JScript FileSystemObject Object DeleteFolder Method

```
object.DeleteFolder ( folderspec[, force] );
```

Arguments: JScript FileSystemObject Object DeleteFolder Method

object

The name of a **FileSystemObject** object. Required.

folderspec

The name of the folder to delete. The *folderspec* can contain wildcard characters in the last path component. Required.

force

A **Boolean** value that is **true** if folders with the read-only attribute set are to be deleted; **false** (default) if they are not. Optional.

Remarks: JScript FileSystemObject Object DeleteFolder Method

The **DeleteFolder** method does not distinguish between folders that have contents and those that do not. The specified folder is deleted regardless of whether or not it has contents.

An error occurs if no matching folders are found. The **DeleteFolder** method stops on the first error it encounters. No attempt is made to roll back or undo any changes that were made before an error occurred.

JScript FileSystemObject Object DriveExists Method

Returns **True** if the specified drive exists; **False** if it does not.

Syntax: JScript FileSystemObject Object DriveExists Method

```
object.DriveExists(drivespec)
```

Arguments: JScript FileSystemObject Object DriveExists Method

object

Required. Always the name of a **FileSystemObject** object.

drivespec

Required. A drive letter or a complete path specification.

Remarks: JScript FileSystemObject Object DriveExists Method

For drives with removable media, the **DriveExists** method returns **True** even if there are no media present. Use the **IsReady** property of the **Drive** object to determine if a drive is ready.

JScript FileSystemObject Object Drives Property

Returns a **Drives** collection consisting of all **Drive** objects available on the local machine.

Syntax: JScript FileSystemObject Object Drives Property

```
object.Drives
```

Arguments: JScript FileSystemObject Object Drives Property

object

A **FileSystemObject**.

Remarks: JScript FileSystemObject Object Drives Property

Removable-media drives need not have media inserted for them to appear in the **Drives** collection.

You can iterate the members of the **Drives** collection using the **Enumerator** object and the **for** statement:

```
function ShowDriveList()

{
```

```
var fs, s, n, e, x;
fs = new ActiveXObject("Scripting.FileSystemObject");
e = new Enumerator(fs.Drives);
s = "";
for (; !e.atEnd(); e.moveNext())
{
x = e.item();
s = s + x.DriveLetter;
s += " - ";
if (x.DriveType == 3)
n = x.ShareName;
else if (x.IsReady)
n = x.VolumeName;
else
n = "[Drive not ready]";
s += n + "<br>";
}
Response.Write(s);
}
```

Under UNIX the **Drives** collection has only one member, "/".

JScript FileSystemObject Object FileExists Method

Returns **True** if a specified file exists, **False** if it does not.

Syntax: JScript FileSystemObject Object FileExists Method

object.**FileExists(***filespec***)**

Arguments: JScript FileSystemObject Object FileExists Method

object

The name of a **FileSystemObject** object. Required.

filespec

The name of the file whose existence is to be determined. A complete path specification (either absolute or relative) must be provided if the file isn't expected to exist in the current folder. Required.

JScript FileSystemObject Object FolderExists Method

Returns **True** if a specified folder exists; **False** if it does not.

Syntax: JScript FileSystemObject Object FolderExists Method

```
object.FolderExists(folderspec)
```

Arguments: JScript FileSystemObject Object FolderExists Method

object

The name of a **FileSystemObject** object. Required.

folderspec

The name of the folder whose existence is to be determined. A complete path specification (either absolute or relative) must be provided if the folder isn't expected to exist in the current folder. Required.

JScript FileSystemObject Object GetAbsolutepathname[0] Method

Returns a complete and unambiguous path from a provided path specification.

Syntax: JScript FileSystemObject Object GetAbsolutepathname Method

```
object.GetAbsolutepathname(pathspec)
```

Arguments: JScript FileSystemObject Object GetAbsolutepathname Method

object

The name of a **FileSystemObject** object. Required.

pathspec

The path specification to change to a complete and unambiguous path. Required.

Remarks: JScript FileSystemObject Object GetAbsolutepathname Method

A path is complete and unambiguous if it provides a complete reference from the root of the specified drive. A complete path can only end with a path separator character (\) if it specifies the root folder of a mapped drive.

Assuming the current directory is c:\mydocuments\reports, the following table illustrates the behavior of the **GetAbsolutepathname** method.

pathspec	Returned path
"c:"	"c:\mydocuments\reports"
"c:.."	"c:\mydocuments"
"c:\\\\\\"	"c:\"
"c:*.*\\may97"	"c:\mydocuments\reports*.*\may97"
"region1"	"c:\mydocuments\reports\region1"

"c:\\..\\..\\mydocuments" "c:\mydocuments"

JScript FileSystemObject Object GetBaseName Method

Returns a string containing the base name of the last component, less any file extension, in a path.

Syntax: JScript FileSystemObject Object GetBaseName Method

```
object.GetBaseName(path)
```

Arguments: JScript FileSystemObject Object GetBaseName Method

object

The name of a **FileSystemObject**/ object. Required.

path

The path specification for the component whose base name is to be returned. Required.

Remarks: JScript FileSystemObject Object GetBaseName Method

The **GetBaseName** method returns a zero-length string ("") if no component matches the *path* argument.

Note

The **GetBaseName** method works only on the provided *path* string. It does not attempt to resolve the path, nor does it check for the existence of the specified path.

JScript FileSystemObject Object GetDrive Method

Returns a **Drive** object corresponding to the drive in a specified path.

Syntax: JScript FileSystemObject Object GetDrive Method

```
object.GetDrive ( drivespec );
```

Arguments: JScript FileSystemObject Object GetDrive Method

object

The name of a **FileSystemObject** object. Required.

drivespec

For Windows, the *drivespec* argument can be a drive letter (c), a drive letter with a colon appended (c:), a drive letter with a colon and path separator appended (c:\), or any network share specification (\\computer2\share1). For UNIX servers the only valid *drivespec* is "/". Required.

Remarks: JScript FileSystemObject Object GetDrive Method

For network shares, a check is made to ensure that the share exists.

An error occurs if *drivespec* does not conform to one of the accepted forms or does not exist.

To call the **GetDrive** method on a normal *path* string, use the following sequence to get a string that is suitable for use as *drivespec*:

```
DriveSpec = GetDriveName(GetAbsolutepathname(Path))
```

Under UNIX, the **GetDrive** method always returns a **Drive** object for '/'.

JScript FileSystemObject Object GetDriveName Method

Returns a string containing the name of the drive for a specified path.

Syntax: JScript FileSystemObject Object GetDriveName Method

```
object.GetDriveName(path)
```

Arguments: JScript FileSystemObject Object GetDriveName Method

object

The name of a **FileSystemObject**. Required.

path

The path specification for the component whose drive name is to be returned. Required.

Remarks: JScript FileSystemObject Object GetDriveName Method

The **GetDriveName** method returns a zero-length string ("") if the drive can't be determined.

On UNIX systems the **GetDriveName** method always returns "/".

Note

> The **GetDriveName** method works only on the provided *path* string. It does not attempt to resolve the path, nor does it check for the existence of the specified path.

JScript FileSystemObject Object GetExtensionName Method

Returns a string containing the extension name for the last component in a path.

Syntax: JScript FileSystemObject Object GetExtensionName Method

```
object.GetExtensionName(path)
```

Arguments: JScript FileSystemObject Object GetExtensionName Method

object

The name of a **FileSystemObject**. Required.

path

The path specification for the component whose extension name is to be returned. Required.

Remarks: JScript FileSystemObject Object GetExtensionName Method

For network drives, the root directory (\) is considered to be a component.

The **GetExtensionName** method returns a zero-length string ("") if no component matches the *path* argument.

JScript FileSystemObject Object GetFile Method

Returns a **File** object corresponding to the file in a specified path.

Syntax: JScript FileSystemObject Object GetFile Method

```
object.GetFile(filespec)
```

Arguments: JScript FileSystemObject Object GetFile Method

object

The name of a **FileSystemObject**. Required.

filespec

The path (absolute or relative) to a specific file. Required.

Remarks: JScript FileSystemObject Object GetFile Method

An error occurs if the specified file does not exist.

JScript FileSystemObject Object GetFileName Method

Returns the last component of specified path that is not part of the drive specification.

Syntax: JScript FileSystemObject Object GetFileName Method

```
object.GetFileName(pathspec)
```

Arguments: JScript FileSystemObject Object GetFileName Method

object

The name of a **FileSystemObject**. Required.

pathspec

The path (absolute or relative) to a specific file. Required.

Remarks: JScript FileSystemObject Object GetFileName Method

The **GetFileName** method returns a zero-length string ("") if *pathspec* does not end with the named component.

Note

The **GetFileName** method works only on the provided *path* string. It does not attempt to resolve the path, nor does it check for the existence of the specified path.

JScript FileSystemObject Object GetFolder Method

Returns a **Folder** object corresponding to the folder in a specified path.

Syntax: JScript FileSystemObject Object GetFolder Method

```
object.GetFolder(folderspec)
```

Arguments: JScript FileSystemObject Object GetFolder Method

object

The name of a **FileSystemObject**. Required.

folderspec

The path (absolute or relative) to a specific folder. Required.

Remarks: JScript FileSystemObject Object GetFolder Method

An error occurs if the specified folder does not exist.

JScript FileSystemObject Object GetParentFolderName Method

Returns a string containing the name of the parent folder of the last component in a specified path.

Syntax: JScript FileSystemObject Object GetParentFolderName Method

```
object.GetParentFolderName(path)
```

Arguments: JScript FileSystemObject Object GetParentFolderName Method

object

The name of a **FileSystemObject**. Required.

path

The path specification for the component whose parent folder name is to be returned. Required.

Remarks: JScript FileSystemObject Object GetParentFolderName Method

The **GetParentFolderName** method returns a zero-length string ("") if there is no parent folder for the component specified in the *path* argument.

Note

> The **GetParentFolderName** method works only on the provided *path* string. It does not attempt to resolve the path, nor does it check for the existence of the specified path.

JScript FileSystemObject Object GetSpecialFolder Method

Returns the special folder specified.

Syntax: JScript FileSystemObject Object GetSpecialFolder Method

```
object.GetSpecialFolder(folderspec)
```

Arguments: JScript FileSystemObject Object GetSpecialFolder Method

object

The name of a **FileSystemObject**. Required.

folderspec

The name of the special folder to be returned. Can be any of the constants shown in the Settings section. Required.

Settings: JScript FileSystemObject Object GetSpecialFolder Method

The *folderspec* argument can have any of the following values:

Constant	Value	Description
WindowsFolder	0	The Windows folder contains files installed by the Windows operating system. *Not available on Unix.*
SystemFolder	1	The System folder contains libraries, fonts, and device drivers. *Not available on Unix.*
TemporaryFolder	2	The Temp folder is used to store temporary files. Its path is found in the TMP environment variable.

JScript FileSystemObject Object GetTempName Method

Returns a randomly generated temporary file or folder name that is useful for performing operations that require a temporary file or folder.

Syntax: JScript FileSystemObject Object GetTempName Method

```
object.GetTempName ( );
```

Arguments: JScript FileSystemObject Object GetTempName Method

object

The name of a **FileSystemObject**. Optional.

Remarks: JScript FileSystemObject Object GetTempName Method

The **GetTempName** method does not create a file. It provides only a temporary file name that can be used with **CreateTextFile** to create a file.

JScript FileSystemObject Object MoveFile Method

Moves one or more files from one location to another.

Syntax: JScript FileSystemObject Object MoveFile Method

```
object.MoveFile ( source, destination );
```

Arguments: JScript FileSystemObject Object MoveFile Method

object

The name of a **FileSystemObject**. Required.

source

The path to the file or files to be moved. The *source* argument string can contain wildcard characters in the last path component only. Required.

destination

The path where the file or files are to be moved. The *destination* argument can't contain wildcard characters. Required.

Remarks: JScript FileSystemObject Object MoveFile Method

If *source* contains wildcards or *destination* ends with a path separator (\), it is assumed that *destination* specifies an existing folder in which to move the matching files. Otherwise, *destination* is assumed to be the name of a destination file to create. In either case, three things can happen when an individual file is moved:

- If *destination* does not exist, the file gets moved. This is the usual case.
- If *destination* is an existing file, an error occurs.
- If *destination* is a directory, an error occurs.

An error also occurs if a wildcard character that is used in *source* doesn't match any files. The **MoveFile** method stops on the first error it encounters. No attempt is made to roll back any changes made before the error occurs.

Important

This method allows moving files between volumes only if supported by the operating system.

JScript FileSystemObject Object MoveFolder Method

Moves one or more folders from one location to another.

Syntax: JScript FileSystemObject Object MoveFolder Method

```
object.MoveFolder ( source, destination );
```

Arguments: JScript FileSystemObject Object MoveFolder Method

object

The name of a **FileSystemObject**. Required.

source

The path to the folder or folders to be moved. The *source* argument string can contain wildcard characters in the last path component only. Required.

destination

The path where the folder or folders are to be moved. The *destination* argument can't contain wildcard characters. Required.

Remarks: JScript FileSystemObject Object MoveFolder Method

If *source* contains wildcards or *destination* ends with a path separator (\), it is assumed that *destination* specifies an existing folder in which to move the matching files. Otherwise, *destination* is assumed to be the name of a destination folder to create. In either case, three things can happen when an individual folder is moved:

- If *destination* does not exist, the folder gets moved. This is the usual case.
- If *destination* is an existing file, an error occurs.
- If *destination* is a directory, an error occurs.

An error also occurs if a wildcard character that is used in *source* doesn't match any folders. The **MoveFolder** method stops on the first error it encounters. No attempt is made to roll back any changes made before the error occurs.

Important

This method allows moving folders between volumes only if supported by the operating system.

JScript FileSystemObject Object OpenTextFile Method

Opens a specified file and returns a **TextStream** object that can be used to read from or append to the file.

Syntax: JScript FileSystemObject Object OpenTextFile Method

```
object.OpenTextFile(filename[, iomode[, create[, format]]])
```

Arguments: JScript FileSystemObject Object OpenTextFile Method

object

The name of a **FileSystemObject**. Required.

filename

A **String** expression that identifies the file to open. Required.

iomode

Indicates input/output mode. Can be one of three constants: **ForReading**, **ForWriting**, or **ForAppending**. Optional.

create

A Boolean value that indicates whether a new file can be created if the specified *filename* doesn't exist. The value is **True** if a new file is to be created, **False** if it isn't to be created. If omitted, a new file isn't created. Optional.

format

One of three **Tristate** values used to indicate the format of the opened file. If omitted, the file is opened as ASCII. Optional.

Settings: JScript FileSystemObject Object OpenTextFile Method

The *iomode* argument can have any of the following settings:

Constant	Value	Description
ForReading	1	Open a file for reading only. You can't write to this file.
ForWriting	2	Open a file for writing. If a file with the same name exists, its previous contents are overwritten.
ForAppending	8	Open a file and write to the end of the file.

The *format* argument can have any of the following settings:

Constant	Value	Description
TristateUseDefault	-2	Opens the file using the system default.
TristateTrue	-1	Opens the file as Unicode.
TristateFalse	0	Opens the file as ASCII.

Remarks: JScript FileSystemObject Object OpenTextFile Method

The following code illustrates the use of the **OpenTextFile** method to open a file for appending text:

```
var fs, a, ForAppending;
ForAppending = 8;
fs = new ActiveXObject("Scripting.FileSystemObject");
a = fs.OpenTextFile("c:\\testfile.txt", ForAppending, false);
...
a.Close();
```

JScript Folder Object

JScript Folder Object

The **Folder** object provides access to all the properties of a folder.

Methods: JScript Folder Object

JScript Folder Object Copy Method	Copies a folder and its contents from one location to another.
JScript Folder Object CreateTextFile Method	Creates a file with a specified name and returns a **TextStream** object.
JScript Folder Object Delete Method	Deletes a folder and its contents.
JScript Folder Object Move Method	Moves a folder and its contents from one location to another.

Properties: JScript Folder Object

JScript Folder Object Attributes Property	The file system attributes of the folder.
JScript Folder Object DateCreated Property	The date and time that a folder was created.
JScript Folder Object DateLastAccessed Property	The date and time that a folder was last accessed.
JScript Folder Object DateLastModified Property	The date and time that a folder was last modified.
JScript Folder Object Drive Property	The drive letter of the drive on which the file resides. *On UNIX, this property is always '/'.*
JScript Folder Object Files Property	A collection of all **File** objects contained in the folder. See the **Files** collection section for details.
JScript Folder Object IsRootFolderProperty	Boolean indicating whether this folder is the root folder of a drive.
JScript Folder Object Name Property	The name of the folder.
JScript Folder Object ParentFolder Property	The **Folder** object that contains this folder.
JScript Folder Object Path Property	The file system path to this folder.
JScript Folder Object ShortName Property	The short name used by programs that require 8.3 file names. *This property is not currently supported on UNIX.*
JScript Folder Object ShortPath Property	The short path used by programs that require 8.3 file names *This property is not currently supported on UNIX.*
JScript Folder Object Size Property	The size, in bytes, of all files and subfolders contained in the folder.
JScript Folder Object SubFolders Property	A collection of all **Folder** objects contained in the folder. See the **Folders** collection for details.
JScript Folder Object Type Property	Information about the type of a folder. *This*

property is not currently supported on UNIX.

Remarks: JScript Folder Object

The following code illustrates how to obtain a **Folder** object and how to return one of its properties:

```
function ShowFolderInfo(folderspec)
{
var fs, folder, s;
fs = new ActiveXObject("Scripting.FileSystemObject");
folder = fs.GetFolder(folderspec);
s = folder.DateCreated;
Response.Write(s);
}
```

JScript Folder Object Attributes Property

Sets or returns the attributes of folders. Read/write or read-only, depending on the attribute.

Note

This property depends on the underlying operating system for its behavior. If the OS file system does not support the folder attribute requested, an error will be returned.

Syntax: JScript Folder Object Attributes Property

```
object.Attributes [= newattributes]
```

Arguments: JScript Folder Object Attributes Property

object

The name of a **Folder** object. Required.

newattributes

If provided, *newattributes* is the new value for the attributes of the specified *object*. Optional.

Settings: JScript Folder Object Attributes Property

The *newattributes* argument can have any of the following values or any logical combination of the following values:

Constant	Value	Description
Normal	0	Normal file. No attributes are set.
ReadOnly	1	Read-only file. Attribute is read/write.
Hidden	2	Hidden file. Attribute is read/write.

System	4	System file. Attribute is read/write.
Volume	8	Disk drive volume label. Attribute is read-only.
Directory	16	Folder or directory. Attribute is read-only.
Archive	32	File has changed since last backup. Attribute is read/write.
Alias	64	Link or shortcut. Attribute is read-only.
Compressed	128	Compressed file. Attribute is read-only.

Remarks: JScript Folder Object Attributes Property

The following code illustrates the use of the **Attributes** property with a folder:

```
function ToggleArchiveBit(filespec)
{
var fs, f, r, s;
fs = new ActiveXObject("Scripting.FileSystemObject");
f = fs.GetFolder(filespec)
if (f.attributes && 32)
{
f.attributes = f.attributes - 32;
s = "Archive bit is cleared.";
}
else
{
f.attributes = f.attributes + 32;
s = "Archive bit is set.";
}
return s;
}
```

JScript Folder Object Copy Method

Copies a specified folder from one location to another.

Syntax: JScript Folder Object Copy Method

```
object.Copy( destination[, overwrite] );
```

Arguments: JScript Folder Object Copy Method

object

The name of a **Folder** object. Required.

destination

The destination where the folder is to be copied. Wildcard characters are not allowed. Required.

overwrite

A **Boolean** value that is **True** (default) if existing folders are to be overwritten; **False** if they are not. Optional.

Remarks: JScript Folder Object Copy Method

The results of the **Copy** method on a **Folder** are identical to operations performed using **CopyFolder** where the folder referred to by *object* is passed as an argument. You should note, however, that the alternative method is capable of copying multiple folders.

JScript Folder Object CreateTextFile Method

Creates a specified file name and returns a **TextStream** object that can be used to read from or write to the file.

Syntax: JScript Folder Object CreateTextFile Method

```
object.CreateTextFile(filename[, overwrite[, unicode]])
```

Arguments: JScript Folder Object CreateTextFile Method

object

The name of a **Folder** object. Required.

filename

A **String** expression that identifies the file to create. Required.

overwrite

A **Boolean** value that indicates whether you can overwrite an existing file. The value is true if the file can be overwritten, false if it can't be overwritten. If omitted, existing files are not overwritten. Optional.

unicode

A **Boolean** value that indicates whether the file is created as a Unicode or ASCII file. The value is true if the file is created as a Unicode file, false if it's created as an ASCII file. If omitted, an ASCII file is assumed. Optional.

JScript Folder Object DateCreated Property

Returns the date and time that the specified folder was created. Read-only.

Syntax: JScript Folder Object DateCreated Property

```
object.DateCreated
```

Arguments: JScript Folder Object DateCreated Property

object

A **File** or **Folder** object.

Remarks: JScript Folder Object DateCreated Property

The following code illustrates the use of the **DateCreated** property with a folder:

```
function ShowFileInfo(filespec)
{
var fs, f, s;
fs = new ActiveXObject("Scripting.FileSystemObject");
f = fs.GetFolder(filespec);
s = "Created: " + f.DateCreated;
Response.Write(s);
}
```

JScript Folder Object DateLastAccessed Property

Returns the date and time that the specified folder was last accessed. Read-only.

Syntax: JScript Folder Object DateLastAccessed Property

```
object.DateLastAccessed
```

Arguments: JScript Folder Object DateLastAccessed Property

object

A **File** or **Folder** object.

Remarks: JScript Folder Object DateLastAccessed Property

The following code illustrates the use of the **DateLastAccessed** property with a folder:

```
function ShowFileAccessInfo(filespec)
{
var fs, f, s;
fs = new ActiveXObject("Scripting.FileSystemObject");
f = fs.GetFolder(filespec);
s = filespec.toUpperCase() + "<br>";
s += "Created: " + f.DateCreated + "<br>";
s += "Last Accessed: " + f.DateLastAccessed + "<br>";
```

```
s += "Last Modified: " + f.DateLastModified;
Response.Write(s);
}
```

Note

This method depends on the underlying operating system for its behavior. If the operating system does not support providing time information, none will be returned.

JScript Folder Object DateLastModified Property

Returns the date and time that the specified folder was last modified. Read-only.

Syntax: JScript Folder Object DateLastModified Property

```
object.DateLastModified
```

Arguments: JScript Folder Object DateLastModified Property

object

A **File** or **Folder** object.

Remarks: JScript Folder Object DateLastModified Property

The following code illustrates the use of the **DateLastModified** property with a folder:

```
function ShowFileAccessInfo(filespec)
{
var fs, f, s;
fs = new ActiveXObject("Scripting.FileSystemObject");
f = fs.GetFolder(filespec);
s = filespec.toUpperCase() + "<br>";
s += "Created: " + f.DateCreated + "<br>";
s += "Last Accessed: " + f.DateLastAccessed + "<br>";
s += "Last Modified: " + f.DateLastModified;
Response.Write(s);
}
```

JScript Folder Object Delete Method

Deletes a specified folder.

Syntax: JScript Folder Object Delete Method

```
object.Delete( force );
```

Arguments: JScript Folder Object Delete Method

object

Required. Always the name of a **Folder** object.

force

Optional. Boolean value that is **True** if folders with the read-only attribute set are to be deleted, **False** (default) if they are not.

Remarks: JScript Folder Object Delete Method

An error occurs if the specified folder does not exist.

The results of the **Delete** method on a **Folder** are identical to operations performed using **DeleteFolder.**

The **Delete** method does not distinguish between folders that have contents and those that do not. The specified folder is deleted regardless of whether or not it has contents.

JScript Folder Object Drive Property

Returns the drive letter of the drive on which the specified folder resides. Read-only.

Syntax: JScript Folder Object Drive Property

```
object.Drive
```

Arguments: JScript Folder Object Drive Property

object

A **Folder** object.

Remarks: JScript Folder Object Drive Property

On UNIX systems, the **Drive** property is always "/".

The following code illustrates the use of the **Drive** property on Windows systems:

```
function ShowFileAccessInfo(filespec)
{
var fs, f, s;
fs = new ActiveXObject("Scripting.FileSystemObject");
f = fs.GetFolder(filespec);
s = f.Name + " on Drive " + f.Drive + "<br>";
s += "Created: " + f.DateCreated + "<br>";
s += "Last Accessed: " + f.DateLastAccessed + "<br>";
s += "Last Modified: " + f.DateLastModified;
Response.Write(s);
```

```
}
```

JScript Folder Object Files Property

Returns a **Files** collection consisting of all **File** objects contained in the specified folder, including those with hidden and system file attributes set.

Syntax: JScript Folder Object Files Property

```
object.Files
```

Arguments: JScript Folder Object Files Property

object

A **Folder** object.

Remarks: JScript Folder Object Files Property

The following code illustrates the use of the **Files** property:

```
function ShowFolderFileList(folderspec)
{
var fs, f, f1, fc, s;
fs = new ActiveXObject("Scripting.FileSystemObject");
f = fs.GetFolder(folderspec);
fc = new Enumerator(f.files);
s = "";
for (; !fc.atEnd(); fc.moveNext())
{
s += fc.item();
s += "<br>";
}
Response.Write(s);
}
```

JScript Folder Object IsRootFolder Property

Returns **True** if the specified folder is the root folder; **False** if it is not.

Syntax: JScript Folder Object IsRootFolder Property

```
object.IsRootFolder
```

Arguments: JScript Folder Object IsRootFolder Property

object

A **Folder** object.

Remarks: JScript Folder Object IsRootFolder Property

The following code illustrates the use of the **IsRootFolder** property:

```
function DisplayLevelDepth(pathspec)
{
var fs, f, n;
fs = new ActiveXObject("Scripting.FileSystemObject");
f = fs.GetFolder(pathspec);
n = 0;
if (f.IsRootFolder)
Response.Write("The specified folder is the root folder.")
else
{
do
{
f = f.ParentFolder;
n++;
}
while (!f.IsRootFolder)
Response.Write("The specified folder is nested " + n + " levels
deep.")
}
}
```

JScript Folder Object Move Method

Moves a specified folder from one location to another.

Syntax: JScript Folder Object Move Method

```
object.Move( destination );
```

Arguments: JScript Folder Object Move Method

object

The name of a **Folder** object. Required.

destination

Destination where the folder is to be moved. Wildcard characters are not allowed. Required.

Remarks: JScript Folder Object Move Method

The results of the **Move** method on a **Folder** are identical to operations performed using **MoveFolder**. You should note, however, that the alternative method is capable of moving multiple folders.

JScript Folder Object Name Property

Sets or returns the name of a specified folder. Read/write.

Syntax: JScript Folder Object Name Property

```
object.Name [= newname]
```

Arguments: JScript Folder Object Name Property

object

The name of a **Folder** object. Required.

newname

The new name of the specified *object*, if provided. Optional.

Remarks: JScript Folder Object Name Property

The following code illustrates the use of the **Name** property:

```
function ShowFileAccessInfo(filespec)
{
var fs, f, s;
fs = new ActiveXObject("Scripting.FileSystemObject");
f = fs.GetFolder(filespec);
s = f.Name + " on Drive " + f.Drive + "<br>";
s += "Created: " + f.DateCreated + "<br>";
s += "Last Accessed: " + f.DateLastAccessed + "<br>";
s += "Last Modified: " + f.DateLastModified;
Response.Write(s);
}
```

JScript Folder Object ParentFolder Property

Returns the folder object for the parent of the specified folder. Read-only.

Syntax: JScript Folder Object ParentFolder Property

```
object.ParentFolder
```

Arguments: JScript Folder Object ParentFolder Property

object

A **Folder** object.

Remarks: JScript Folder Object ParentFolder Property

The following code illustrates the use of the **ParentFolder** property with a folder:

```
function ShowFileAccessInfo(filespec)
{
var fs, f, s;
fs = new ActiveXObject("Scripting.FileSystemObject");
f = fs.GetFolder(filespec);
s = f.Name + " in " + f.ParentFolder + "<br>";
s += "Created: " + f.DateCreated + "<br>";
s += "Last Accessed: " + f.DateLastAccessed + "<br>";
s += "Last Modified: " + f.DateLastModified;
Response.Write(s);
}
```

JScript Folder Object Path Property

Returns the path for a specified folder.

Syntax: JScript Folder Object Path Property

```
object.Path
```

Arguments: JScript Folder Object Path Property

object

A **Folder** object.

JScript Folder Object ShortName Property

Returns the short name used by programs that require the earlier 8.3 naming convention. *This property is not available under UNIX.*

Syntax: JScript Folder Object ShortName Property

```
object.ShortName
```

Arguments: JScript Folder Object ShortName Property

object

A **Folder** object.

Remarks: JScript Folder Object ShortName Property

The following code illustrates the use of the **ShortName** property with a **Folder** object:

```
function ShowShortName(filespec)
{
var fs, f, s;
fs = new ActiveXObject("Scripting.FileSystemObject");
f = fs.GetFolder(filespec);
s = "The short name for " + "" + f.Name;
s += "" + "<br>";
s += "is: " + "" + f.ShortName + "";
Response.Write(s);
}
```

JScript Folder Object ShortPath Property

Returns the short path used by programs that require the earlier 8.3 file naming convention. *This property is not available under UNIX.*

Syntax: JScript Folder Object ShortPath Property

```
object.ShortPath
```

Arguments: JScript Folder Object ShortPath Property

object

A **Folder** object.

Remarks: JScript Folder Object ShortPath Property

The following code illustrates the use of the **ShortPath** property with a **Folder** object:

```
function ShowShortPath(filespec);
{
var fs, f, s;
fs = new ActiveXObject("Scripting.FileSystemObject");
f = fs.GetFolder(filespec);
s = "The short path for " + "" + f.Name;
```

```
s += "" + "<br>";
s += "is: " + "" + f.ShortPath + "";
Response.Write(s);
}
```

JScript Folder Object Size Property

The size, in bytes, of all files and subfolders contained in the folder.

Syntax: JScript Folder Object Size Property

```
object.Size
```

Arguments: JScript Folder Object Size Property

object

A **Folder** object.

Remarks: JScript Folder Object Size Property

The following code illustrates the use of the **Size** property with a **Folder** object:

```
function ShowFolderSize(filespec)
{
var fs, f, s;
fs = new ActiveXObject("Scripting.FileSystemObject");
f = fs.GetFolder(filespec);
s = f.Name + " uses " + f.size + " bytes.";
Response.Write(s);
}
```

JScript Folder Object SubFolders Property

Returns a **Folders** collection consisting of all folders contained in a specified folder, including those with Hidden and System file attributes set.

Syntax: JScript Folder Object SubFolders Property

```
object.SubFolders
```

Arguments: JScript Folder Object SubFolders Property

object

A **Folder** object.

Remarks: JScript Folder Object SubFolders Property

The following code illustrates the use of the **SubFolders** property:

```
function ShowFolderList(folderspec)
{
var fs, f, f1, fc, s;
fs = new ActiveXObject("Scripting.FileSystemObject");
f = fs.GetFolder(folderspec);
fc = new Enumerator(f.SubFolders);
s = "";
for (;!fc.atEnd(); fc.moveNext())
{
s += fc.item();
s += "<br>";
}
Response.Write(s);
}
```

JScript Folder Object Type Property

Returns information about the type of a folder. *This property is not available under UNIX.*

Syntax: JScript Folder Object Type Property

```
object.Type
```

Arguments: JScript Folder Object Type Property

object

A **Folder** object.

Remarks: JScript Folder Object Type Property

The following code illustrates the use of the **Type** property to return a folder type. In this example, try providing the path of the Recycle Bin or other unique folder to the procedure.

```
function ShowFileType(filespec)
{
var fs, f, s;
fs = new ActiveXObject("Scripting.FileSystemObject");
if (fs.FolderExists(filespec))
f = fs.GetFolder(filespec);
```

```
else if (fs.FileExists(filespec))
f = fs.GetFile(filespec);
else
s = "File or Folder does not exist.";
s = f.Name + " is a " + f.Type;
Response.Write(s);
}
```

JScript Function Object

JScript Function Object

The **Function** object creates a new function.

Properties: JScript Function Object

JScript Function Object arguments Property	An array containing each argument passed to the currently executing function.
JScript Function Object caller Property	**A reference to the function that invoked the current function.**
JScript Function Object length Property	The number of arguments to the function.

Syntax 1: JScript Function Object

```
function functionname( [argname1 [, ... argnameN]] )
{
body
}
```

Syntax 2: JScript Function Object

```
var functionname = new Function( [argname1, [... argnameN,]] body );
```

Arguments: JScript Function Object

functionname

The name of the newly created function

argname1...argnameN

An optional list of arguments that the function accepts.

body

A string that contains the block of JScript code to be executed when the function is called.

Remarks: JScript Function Object

Syntax 1 is the standard way to create new functions in JScript. Syntax 2 is an alternative form used to create **Function** objects explicitly.

The function is a basic data type in JScript. Syntax 1 creates a function value that JScript converts into a **Function** object when necessary. JScript converts **Function** objects created by Syntax 2 into function values at the time the function is called.

For example, if you want to create a function that adds the two arguments passed to it, you can do it in either of two ways:

Example 1

```
function add(x, y)
{
return x + y;
}
```

Example 2

```
var add = new Function("x", "y", "return x+y");
```

In either case, you call the function with a line of code similar to the following:

```
add(2, 3);
```

JScript Function Object arguments Property

An array containing each argument passed to the currently executing function.

Syntax: JScript Function Object arguments Property

```
function.arguments[ ]
```

Arguments: JScript Function Object arguments Property

function

The name of the currently executing function.

Remarks: JScript Function Object arguments Property

The **arguments** property allows a graceful way for functions to handle a variable number of arguments. The **length** property of the array contains the number of arguments passed to the function.

JScript Function Object caller Property

Contains a reference to the function that invoked the current function.

Syntax: JScript Function Object caller Property

```
functionname.caller
```

Arguments: JScript Function Object caller Property

functionname

The name of the currently executing function.

Remarks: JScript Function Object caller Property

The **caller** property is only defined for a function while that function is executing. If the function is called from the top level of a JScript program, **caller** contains **null**.

JScript Function Object length Property

Contains the number of arguments a function is defined with.

Syntax: JScript Function Object length Property

```
functionname.length
```

Arguments: JScript Function Object length Property

functionname

The name of the function.

Remarks: JScript Function Object length Property

The **length** property of a function is initialized by the scripting engine to the number of arguments in the function's definition when an instance of the function is created.

What happens when a function is called with a number of arguments different from the value of its **length** property depends on the function.

JScript Global Object

JScript Global Object

The **Global** object is an intrinsic object whose purpose is to collect global methods into one object.

Methods: JScript Global Object

JScript Global Object escape Method	Encodes **String** objects so that they can be read on all computers.
JScript Global Object eval Method	Evaluates JScript code.
JScript Global Object isFinite Method	Determines if a supplied number is finite.
JScript Global Object isNaN Method	Determines is a supplied number is the reserved value **NaN** (Not a Number).
JScript Global Object parseFloat Method	Converts strings to floating point numbers.
JScript Global Object parseInt Method	Converts strings to integer numbers.

JScript Global Object unescape Method	Decodes **String** objects encoded with the **escape** method.

Properties: JScript Global Object

JScript Global Object Infinity Property	An initial value of **Number.POSITIVE_INFINITY.**
JScript Global Object NaN Property	An initial value of **Number.NaN**.

Syntax: JScript Global Object

The **Global** object has no syntax. You call its methods directly.

Remarks: JScript Global Object

The **Global** object is never used directly, and cannot be created using the **new** operator. It is created when the scripting engine is initialized, thus making its methods and properties available immediately.

JScript Global Object escape Method

Encodes **String** objects so they can be read on all computers.

Syntax: JScript Global Object escape Method

```
escape(charstring)
```

Arguments: JScript Global Object escape Method

charstring

A **String** object to be encoded.

Remarks: JScript Global Object escape Method

The **escape** method returns a new **String** object (in Unicode format) that contains the contents of *charstring*. All spaces, punctuation, accented characters, and any other non-ASCII characters are replaced with **%***xx* encoding, where *xx* is equivalent to the hexadecimal number representing the character. For example, a space is returned as "%20."

Characters with a value greater than 255 are stored using the **%u***xxxx* format.

JScript Global Object eval Method

Evaluates JScript code.

Syntax: JScript Global Object eval Method

```
eval(codestring)
```

Arguments: JScript Global Object eval Method

codestring

A **String** object that contains valid JScript code. This string is parsed by the JScript parser and executed.

Remarks: JScript Global Object eval Method

The **eval** function allows dynamic execution of JScript source code. For example, the following code creates a new variable *mydate* that contains a **Date** object:

```
eval("var mydate = new Date();");
```

The code passed to the **eval** method is executed in the same context as the call to the **eval** method.

JScript Global Object Infinity Property

Contains an initial value of **Number.POSITIVE_INFINITY**.

Syntax: JScript Global Object Infinity Property

```
Infinity
```

Remarks: JScript Global Object Infinity Property

The **Infinity** property is a member of the **Global** object, and is made available when the scripting engine is initialized.

JScript Global Object isFinite Method

Determines if a supplied number is finite.

Syntax: JScript Global Object isFinite Method

```
isFinite(number)
```

Arguments: JScript Global Object isFinite Method

number

Required numeric value.

Remarks: JScript Global Object isFinite Method

The **isFinite** method returns **true** if *number* is any value other than **NaN**, negative infinity, or positive infinity. In those three cases, it returns **false**.

JScript Global Object isNaN Method

Determines whether a value is the reserved value **NaN** (not a number).

Syntax: JScript Global Object isNaN Method

```
isNaN(numvalue)
```

Arguments: JScript Global Object isNaN Method

numvalue

The value to be tested against **NaN**.

Remarks: JScript Global Object isNaN Method

The **isNaN** function returns **True** if the value is **NaN**, and **false** otherwise. You typically use this function to test return values from the **parseInt** and **parseFloat** methods.

Alternatively, a variable could be compared to itself. If it compares as unequal, it is **NaN**. This is because **NaN** is the only value which is not equal to itself.

JScript Global Object NaN Property

Contains an initial value of **NaN**.

Syntax: JScript Global Object NaN Property

```
NaN
```

Remarks: JScript Global Object NaN Property

The **NaN** property (not a number) is a member of the **Global** object, and is made available when the scripting engine is initialized.

JScript Global Object parseFloat Method

Converts strings into floating point numbers.

Syntax: JScript Global Object parseFloat Method

```
parseFloat(numstring)
```

Arguments: JScript Global Object parseFloat Method

numstring

A string that contains a floating point number.

Remarks: JScript Global Object parseFloat Method

The **parseFloat** method returns an numerical value equal to the number contained in *numstring*. If no prefix of *numstring* can be successfully parsed into a floating-point number, **NaN** (not a number) is returned.

```
parseFloat("abc") // Returns NaN.

parseFloat("1.2abc") // Returns 1.2.
```

You can test for **NaN** using the **isNaN** method.

JScript Global Object parseInt Method

Converts strings into integers.

Syntax: JScript Global Object parseInt Method

```
parseInt(numstring, [radix])
```

Arguments: JScript Global Object parseInt Method

numstring

A string to convert into a number. Required.

radix

A value between 2 and 36 indicating the base of the number contained in *numstring*. If not supplied, strings with a prefix of '0x' are considered hexidecimal and strings with a prefix of '0' are considered octal. All other strings are considered decimal. Optional.

Remarks: JScript Global Object parseInt Method

The **parseInt** method returns an integer value equal to the number contained in *numstring*. If no prefix of *numstring* can be successfully parsed into an integer, **NaN** (not a number) is returned.

```
parseInt("abc") // Returns NaN.

parseInt("12abc") // Returns 12.
```

You can test for **NaN** using the **isNaN** method.

JScript Global Object unescape Method

Decodes **String** objects encoded with the **escape** method.

Syntax: JScript Global Object unescape Method

```
unescape(charstring)
```

Arguments: JScript Global Object unescape Method

charstring

A **String** object to be decoded.

Remarks: JScript Global Object unescape Method

The **unescape** method returns a new **String** object that contains the contents of *charstring*. All characters encoded with the **%***xx* hexadecimal form are replaced by their ASCII character set equivalents.

Characters encoded in **%u***xxxx* format (Unicode characters) are replaced with the Unicode character with hexadecimal encoding *xxxx*.

JScript Math Object

JScript Math Object

The **Math** object is a built-in object that provides basic mathematics functionality and constants.

Methods: JScript Math Object

JScript Math Object abs Method	Determines the absolute value of its numeric argument.
JScript Math Object acos Method	Computes the arccosine of its numeric argument.
JScript Math Object asin Method	Computes the arcsine of its numeric argument.
JScript Math Object atan Method	Computes the arctangent of its numeric argument.
JScript Math Object atan2 Method	Returns the angle (in radians) from the X axis to a point.
JScript Math Object ceil Method	Determines the smallest integer greater than or equal to its numeric argument.
JScript Math Object cos Method	Computes the cosine of its numeric argument.
JScript Math Object exp Method	Computes *e* to the power of the numeric argument.
JScript Math Object floor Method	Determines the greatest integer that is less than or equal to its numeric argument.
JScript Math Object log Method	Computes the natural logarithm of its numeric argument.
JScript Math Object max Method	Returns the greater of two numeric expressions.
JScript Math Object min Method	Returns the lesser of two numeric expressions.
JScript Math Object pow Method	Returns the value of a base expression taken to a specified power.
JScript Math Object random Method	Returns a pseudo-random number.
JScript Math Object round Method	Rounds the supplied numeric expression to the nearest integer.
JScript Math Object sin Method	Returns the sin of its numeric argument.
JScript Math Object sqrt Method	Returns the square root of a number.
JScript Math Object tan Method	Returns the tangent of its numeric argument.

Properties: JScript Math Object

JScript Math Object E Property	Euler’s constant.
JScript Math Object LN2 Property	The natural logarithm of 2.
JScript Math Object LN10 Property	The natural logarithm of 10.
JScript Math Object LOG2E Property	The base 2 logarithm of *e*.
JScript Math Object LOG10E Property	The base 10 logarithm of *e*.
JScript Math Object PI Property	The ratio of the circumference of a circle to its

	diameter.
JScript Math Object SQRT1_2 Property	The square root of 0.5, or 1 divided by the square root of 2.
JScript Math Object SQRT2 Property	The square root of 2.

Syntax: JScript Math Object

```
Math.[{property | method}]
```

Arguments: JScript Math Object

property

Name of **Math** object property.

method

Name of **Math** object method.

Remarks: JScript Math Object

The **Math** object cannot be created using the **new** operator, and gives an error if you attempt to do so. It is created by the scripting engine when the engine is loaded. All of its methods and properties are available to your script at all times.

JScript Math Object abs Method

Determines the absolute value of its numeric argument.

Syntax: JScript Math Object abs Method

```
Math.abs(number)
```

Arguments: JScript Math Object abs Method

number

A numeric expression for which the absolute value is sought.

JScript Math Object acos Method

Computes the arccosine of its numeric argument.

Syntax: JScript Math Object acos Method

```
Math.acos(number)
```

Arguments: JScript Math Object acos Method

number

A numeric expression for which the arccosine is sought.

JScript Math Object asin Method

Computes the arcsine of its numeric argument.

Syntax: JScript Math Object asin Method

```
Math.asin(number)
```

Arguments: JScript Math Object asin Method

number

A numeric expression for which the arcsine is sought.

JScript Math Object atan Method

Computes the arctangent of its numeric argument.

Syntax: JScript Math Object atan Method

```
Math.atan(number)
```

Arguments: JScript Math Object atan Method

number

A numeric expression for which the arctangent is sought.

JScript Math Object atan2 Method

Returns the angle (in radians) from the X axis to a point (*y*,*x*).

Syntax: JScript Math Object atan2 Method

```
Math.atan2(y, x)
```

Arguments: JScript Math Object atan2 Method

x

A numeric expression representing the cartesian x-coordinate. Required.

y

A numeric expression representing the cartesian y-coordinate. Required.

Remarks: JScript Math Object atan2 Method

The return value is between -*pi* and *pi*, representing the angle of the supplied (*y*,*x*) point.

JScript Math Object ceil Method

Determines the smallest integer greater than or equal to its numeric argument.

Syntax: JScript Math Object ceil Method

```
Math.ceil(number)
```

Arguments: JScript Math Object ceil Method

number

A numeric expression.

JScript Math Object cos Method

Computes the cosine of its numeric argument.

Syntax: JScript Math Object cos Method

```
Math.cos(number)
```

Arguments: JScript Math Object cos Method

number

A numeric expression for which the cosine is sought.

JScript Math Object E Property

Euler's constant, the base of natural algorithms. The **E** property is approximately equal to 2.718.

Example: JScript Math Object E Property

```
var numVar

numVar = Math.E
```

JScript Math Object exp Method

Computes *e* to the power of the supplied numeric argument.

Syntax: JScript Math Object exp Method

```
Math.exp(number)
```

Arguments: JScript Math Object exp Method

number

A numeric expression representing the power of *e*.

Remarks: JScript Math Object exp Method

The return value is e^{number}. The constant *e* is Euler's constant, approximately equal to 2.178 and *number* is the supplied argument.

JScript Math Object floor Method

Computes the greatest integer less than or equal to its numeric argument.

Syntax: JScript Math Object floor Method

```
Math.floor(number)
```

Arguments: JScript Math Object floor Method

number

A numeric expression.

JScript Math Object LN2 Property

The natural logarithm of 2.

Applies To: JScript Math Object LN2 Property

Math

Syntax: JScript Math Object LN2 Property

```
var numVar
numVar = Math.LN2
```

Remarks: JScript Math Object LN2 Property

The **LN2** property is approximately equal to 0.693.

JScript Math Object LN10 Property

The natural logarithm of 10.

Applies To: JScript Math Object LN10 Property

Math

Syntax: JScript Math Object LN10 Property

```
var numVar
numVar = Math.LN10
```

Remarks: JScript Math Object LN10 Property

The **LN10** property is approximately equal to 2.302.

JScript Math Object log Method

Computes the natural logarithm of a numeric expression.

Applies To: JScript Math Object log Method

Math

Syntax: JScript Math Object log Method

```
Math.log(number)
```

Arguments: JScript Math Object log Method

number

A numeric expression for which the natural logarithm is sought.

Return Value: JScript Math Object log Method

The return value is the natural logarithm of *number*. The base is *e*.

JScript Math Object LOG2E Property

The base 2 logarithm of *e*, Euler's constant.

Applies To: JScript Math Object LOG2E Property

Math

Syntax: JScript Math Object LOG2E Property

```
var varName
varName = Math.LOG2E
```

Remarks: JScript Math Object LOG2E Property

The **LOG2E** property, a constant, is approximately equal to 1.442.

JScript Math Object LOG10E Property

The base 10 logarithm of *e*, Euler's constant.

Applies To: JScript Math Object LOG10E Property

Math

Syntax: JScript Math Object LOG10E Property

```
var varName
varName = objName.LOG10E
```

Remarks: JScript Math Object LOG10E Property

The **LOG10E** property, a constant, is approximately equal to 0.434.

JScript Math Object max Method

Returns the greater of two supplied numeric expressions.

Syntax: JScript Math Object max Method

```
retVal = Math.max(number1, number2)
```

Arguments: JScript Math Object max Method

retVal

The greater of *number1* or *number2*.

number1

A numeric expression to be compared to *number2*.

number2

A numeric value to be compared to *number1*.

JScript Math Object min Method

Returns the lesser of two supplied numbers.

Syntax: JScript Math Object min Method

```
retVal = Math.min(number1, number2)
```

Arguments: JScript Math Object min Method

retVal

The lesser of *number1* or *number2*.

number1

A numeric expression to be compared to *number2*.

number2

A numeric value to be compared to *number1*.

JScript Math Object PI Property

The ratio of the circumference of a circle to its diameter.

Syntax: JScript Math Object PI Property

```
var numVar

numVar = Math.PI
```

Remarks: JScript Math Object PI Property

The **PI** property, a constant, is approximately equal to 3.14159.

JScript Math Object pow Method

Returns the value of a base expression taken to a specified power.

Syntax: JScript Math Object pow Method

```
Math.pow(base, exponent)
```

Arguments: JScript Math Object pow Method

base

The base value of the expression.

exponent

The exponent value of the expression.

Remarks: JScript Math Object pow Method

In the following example, a numeric expression equal to *baseexponent* returns 1000.

```
Math.pow(10,3);
```

JScript Math Object random Method

Returns a pseudorandom number between 0 and 1.

Syntax: JScript Math Object random Method

```
Math.random( )
```

Remarks: JScript Math Object random Method

The pseudorandom number generated is between 0 and 1 inclusive. The random number generator is seeded automatically when JScript is first loaded.

JScript Math Object round Method

Rounds a supplied numeric expression to the nearest integer.

Syntax: JScript Math Object round Method

```
Math.round(number)
```

Arguments: JScript Math Object round Method

number

The value to be rounded to the nearest integer.

Remarks: JScript Math Object round Method

If the decimal portion of *number* is 0.5 or greater, the return value is equal to the smallest integer greater than *number*. Otherwise, **round** returns the largest integer less than or equal to *number*.

JScript Math Object sin Method

Returns the sine of its numeric argument.

Syntax: JScript Math Object sin Method

```
Math.sin(number)
```

Arguments: JScript Math Object sin Method

number

A numeric expression for which the sine is sought.

JScript Math Object sqrt Method

Returns the square root of a number.

Syntax: JScript Math Object sqrt Method

```
Math.sqrt(number)
```

Arguments: JScript Math Object sqrt Method

number

A numeric expression.

Remarks: JScript Math Object sqrt Method

If *number* is negative, the return value is zero.

JScript Math Object SQRT1_2 Property

The square root of 0.5, or 1 divided by the square root of 2.

Syntax: JScript Math Object SQRT1_2 Property

```
var numVar
numVar = Math.SQRT1_2
```

Remarks: JScript Math Object SQRT1_2 Property

The **SQRT1_2** property, a constant, is approximately equal to 0.707.

JScript Math Object SQRT2 Property

The square root of 2.

Syntax: JScript Math Object SQRT2 Property

```
var numVar
numVar = Math.SQRT2
```

Remarks: JScript Math Object SQRT2 Property

The **SQRT2** property, a constant, is approximately equal to 1.414.

JScript Math Object tan Method

Computes the tangent of a number.

Syntax: JScript Math Object tan Method

```
Math.tan(number)
```

Arguments: JScript Math Object tan Method

number

A numeric expression for which the tangent is sought.

JScript Number Object

JScript Number Object

The **Number** object is an object representation of the number data type and placeholder for numeric constants.

Methods: JScript Number Object

JScript Number Object toString Method	Converts a **Number** object or variable to a string using the specified radix.

Properties: JScript Number Object

JScript Number Object MAX_VALUE Property	The largest number that can be represented in JScript.
JScript Number Object MIN_VALUE Property	The number closest to zero that can be represented in JScript.
JScript Number Object NaN Property	A special value that indicates an expression returned a value that was not a number.
JScript Number Object NEGATIVE_INFINITY Property	A value larger than the largest negative number that can be represented in JScript.
JScript Number Object POSITIVE_INFINITY Property	A value larger than the largest positive number that can be represented in JScript.

Syntax: JScript Number Object

```
new Number( value )
```

Arguments: JScript Number Object

value

is the sought numerical value for the object.

Remarks: JScript Number Object

JScript creates **Number** objects as required from numerical values. It is rarely necessary to create **Number** objects explicitly.

The primary purposes for the **Number** object are to collect its properties into one object, and to allow numbers to be converted into strings via the **toString** method.

JScript Number Object MAX_VALUE Property

The largest number that can be represented in JScript. Equal to approximately 1.79E+308.

Syntax: JScript Number Object MAX_VALUE Property

```
Number.MAX_VALUE
```

Arguments: JScript Number Object MAX_VALUE Property

number

The **Number** object.

Remarks: JScript Number Object MAX_VALUE Property

The **Number** object does not have to be created before the **MAX_VALUE** property can be accessed.

JScript Number Object MIN_VALUE Property

The number closest to zero that can be represented in JScript. Equal to approximately 2.22E-308.

Syntax: JScript Number Object MIN_VALUE Property

```
Number.MIN_VALUE
```

Arguments: JScript Number Object MIN_VALUE Property

number

The **Number** object.

Remarks: JScript Number Object MIN_VALUE Property

The **Number** object does not have to be created before the **MIN_VALUE** property can be accessed.

JScript Number Object NaN Property

A special value that indicates an arithmetic expression returned a value that was not a number.

Syntax: JScript Number Object NaN Property

```
Number.NaN
```

Arguments: JScript Number Object NaN Property

number

The **Number** object.

Remarks: JScript Number Object NaN Property

The **Number** object does not have to be created before the **NaN** property can be accessed.

NaN does not compare equal to any value, including itself. To test if a value is equivalent to **NaN**, use the **isNaN** function.

JScript Number Object NEGATIVE_INFINITY Property

A value more negative than the largest negative number (**-Number.MAX_VALUE**) that can be represented in JScript.

Syntax: JScript Number Object NEGATIVE_INFINITY Property

```
Number.NEGATIVE_INFINITY
```

Arguments: JScript Number Object NEGATIVE_INFINITY Property

number

The **Number** object.

Remarks: JScript Number Object NEGATIVE_INFINITY Property

The **Number** object does not have to be created before the **NEGATIVE_INFINITY** property can be accessed.

JScript displays **NEGATIVE_INFINITY** values as -infinity. This value behaves mathematically as infinity.

JScript Number Object POSITIVE_INFINITY Property

A value larger than the largest number (**Number.MAX_VALUE**) that can be represented in JScript.

Syntax: JScript Number Object POSITIVE_INFINITY Property

```
Number.POSITIVE_INFINITY
```

Arguments: JScript Number Object POSITIVE_INFINITY Property

number

The **Number** object.

Remarks: JScript Number Object POSITIVE_INFINITY Property

The **Number** object does not have to be created before the **POSITIVE_INFINITY** property can be accessed.

JScript displays **POSITIVE_INFINITY** values as infinity. This value behaves mathematically as infinity.

JScript Number Object toString Method

Converts a **Number** object or variable into a string using the specified radix.

Syntax: JScript Number Object toString Method

```
number.toString([radix])
```

Arguments: JScript Number Object toString Method

number

A **Number** object or variable. Cannot be a numeric literal

radix

An optional argument specifying the base to use when converting the number. Values between 2 and 16 inclusive are accepted. If not supplied, base 10 is used.

Remarks: JScript Number Object toString Method

Due to syntax restrictions, the **toString** method cannot be used on numeric literals. Values must be assigned to variables first. For example, the following code is invalid:

```
str=1997.toString();
```

This code must be rewritten:

```
var yr = 1997
str = yr.toString()
```

JScript RegExp Object

JScript RegExp Object

The **RegExp** object stores information on regular expression pattern searches.

Properties: JScript RegExp Object

JScript RegExp Object $1. . . $9 Property	The nine most-recently used memorized portions found during pattern matching.
JScript RegExp Object index Property	The location where the first successful match begins.
JScript RegExp Object input Property	The string against which a search was performed.
JScript RegExp Object lastIndex Property	Where the last successful match begins.
JScript RegExp Object lastMatch Property	The last matched characters.
JScript RegExp Object lastParen Property	The last parenthesized substring match, if any.
JScript RegExp Object leftContext Property	The input string up to the most recent match.
JScript RegExp Object multiline Property	Indicates whether searching continued across line breaks.
JScript RegExp Object rightContext Property	The input string past the most recent match.

Syntax: JScript RegExp Object

```
RegExp.propertyname
```

Arguments: JScript RegExp Object

propertyname

One of the **RegExp** object properties.

Remarks: JScript RegExp Object

The **RegExp** object cannot be created directly, but is always available for use. Its properties have undefined as their value until a successful regular expression search has been completed.

JScript RegExp Object $1. . . $9 Property

Specifies the nine most-recently memorized portions found during pattern matching. Read-only.

Syntax: JScript RegExp Object $1. . . $9 Property

```
RegExp.$n
```

Arguments: JScript RegExp Object $1. . . $9 Property

n

A number between 1 and 9.

Remarks: JScript RegExp Object $1. . . $9 Property

The value of the **$1...$9** property is modified whenever a successful parenthesized match is made. Any number of parenthesized substrings may be specified in a regular expression pattern, but only the nine most recent can be stored.

JScript RegExp Object index Property

Indicates where the first successful match begins in a string that was searched.

Syntax: JScript RegExp Object index Property

```
RegExp.index
```

Arguments: JScript RegExp Object index Property

RegExp

A **RegExp** object.

Remarks: JScript RegExp Object index Property

The **index** property is zero-based. Its value is modified whenever a successful match is made.

JScript RegExp Object input Property

Contains the string against which a search was performed. Read-only.

Syntax: JScript RegExp Object input Property

```
RegExp.input
```

Short Syntax: JScript RegExp Object input Property

```
RegExp.$_
```

Arguments: JScript RegExp Object input Property

RegExp

A **RegExp** object.

Remarks: JScript RegExp Object input Property

The value of the **input** property is modified whenever a successful match is made.

JScript RegExp Object lastIndex Property

Indicates where the last successful match begins in a string that was searched.

Syntax: JScript RegExp Object lastIndex Property

```
RegExp.lastIndex
```

Arguments: JScript RegExp Object lastIndex Property

RegExp

The name of the **RegExp** object.

Remarks: JScript RegExp Object lastIndex Property

The **lastIndex** property is zero-based. Its value is modified whenever a successful match is made.

JScript RegExp Object lastMatch Property

Specifies the last matched characters. Read-only.

Syntax: JScript RegExp Object lastMatch Property

```
RegExp.lastMatch
```

Short Syntax: JScript RegExp Object lastMatch Property

```
RegExp.$&
```

Remarks: JScript RegExp Object lastMatch Property

The value of the **lastMatch** property is modified whenever a successful match is made.

JScript RegExp Object lastParen Property

Specifies the last parenthesized substring match, if any. Read-only.

Syntax: JScript RegExp Object lastParen Property

```
RegExp.lastParen
```

Short Syntax: JScript RegExp Object lastParen Property

```
RegExp.$+
```

Remarks: JScript RegExp Object lastParen Property

The value of the **lastParen** property is modified whenever a successful parenthesized match is made.

JScript RegExp Object leftContext Property

Specifies the input string up to the most recent match. Read-only.

Syntax: JScript RegExp Object leftContext Property

```
RegExp.leftContext
```

Short Syntax: JScript RegExp Object leftContext Property

```
RegExp.$`
```

Remarks: JScript RegExp Object leftContext Property

The value of the **leftContext** property is modified whenever a successful match is made.

JScript RegExp Object multiline Property

Indicates whether searching continued across line breaks. Read-only.

Syntax: JScript RegExp Object multiline Property

```
RegExp.multiline
```

Short Syntax: JScript RegExp Object multiline Property

```
RegExp.$*
```

Remarks: JScript RegExp Object multiline Property

The value of the **multiline** property is modified whenever a successful match is made. If **multiline** is **True**, searching was performed across line breaks; otherwise, **multiline** returns **False**.

JScript RegExp Object rightContext Property

Specifies the input string past the most recent match. Read-only.

Syntax: JScript RegExp Object rightContext Property

```
RegExp.rightContext
```

Short Syntax: JScript RegExp Object rightContext Property

```
RegExp.$'
```

Remarks: JScript RegExp Object rightContext Property

The value of the **rightContext** property is modified whenever a successful match is made.

JScript Regular Expression Object

JScript Regular Expression Object

The **Regular Expression** object contains a regular expression pattern.

Methods: JScript Regular Expression Object

JScript Regular Expression Object compile Method	Compiles a regular expression into an internal format.
JScript Regular Expression Object exec Method	Executes a search for a match in a specified string.
JScript Regular Expression Object test Method	Tests whether a pattern exists in a string.

Properties: JScript Regular Expression Object

JScript Regular Expression Object global Property	Indicates whether the global switch (g) has been used.
JScript Regular Expression Object ignoreCase Property	Indicates whether the ignore case switch (i) has been used.
JScript Regular Expression Object lastIndex Property	The index to start the next match.
JScript Regular Expression Object source Property	The text of the regular expression pattern.

Syntax 1: JScript Regular Expression Object

```
var regularexpression = /pattern/[switch]
```

Syntax 2: JScript Regular Expression Object

```
var regularexpression = new RegExp("pattern",["switch"])
```

Arguments: JScript Regular Expression Object

pattern

The regular expression pattern to use. If you use Syntax 1, delimit the pattern by "/" characters. If you use Syntax 2, enclose the pattern in quotes. Required.

switch

Enclose *switch* in quotes if you use Syntax 2. Available switches are: i (ignore case) g (global search for all occurrences of *pattern*) gi (global search, ignore case). Optional.

Remarks: JScript Regular Expression Object

Regular Expression objects store patterns used when searching strings for character combinations. After the **Regular Expression** object is created, it is either passed to a string method, or a string is passed to one of the regular expression methods. Information about the most recent search performed is stored in the **RegExp** object.

Use Syntax 1 when you know the search string ahead of time. Use Syntax 2 when the search string is changing frequently, or is unknown, such as strings taken from user input.

The *pattern* argument is compiled into an internal format before use. For Syntax 1, *pattern* is compiled as the script is loaded. For Syntax 2, *pattern* is compiled just before use, or when the **compile** method is called.

JScript Regular Expression Object compile Method

Compiles a regular expression into an internal format.

Syntax: JScript Regular Expression Object compile Method

```
rgexp.compile(pattern)
```

Arguments: JScript Regular Expression Object compile Method

rgexp

A **RegularExpression** object. Can be a variable name or a literal. Required.

pattern

A **String** expression containing a regular expression pattern to be compiled. Required.

Remarks: JScript Regular Expression Object compile Method

The **compile** method converts *pattern* into an internal format for faster execution. This allows for more efficient use of regular expressions in loops, for example.

JScript Regular Expression Object exec Method

Executes a search for a match in a specified string.

Syntax: JScript Regular Expression Object exec Method

```
rgexp.exec(str)
```

Arguments: JScript Regular Expression Object exec Method

rgexp

A **RegularExpression** object. Can be a variable name or a literal. Required.

str

The string to perform a search on. Required.

Remarks: JScript Regular Expression Object exec Method

The results of an **exec** method search are placed into an array.

If the **exec** method does not find a match, it returns **null**. If it finds one or more matches, the **exec** method returns an array, and the **RegExp** object is updated to reflect the results of the search.

JScript Regular Expression Object global Property

Indicates whether the global switch (g) has been used. Read-only.

Syntax: JScript Regular Expression Object global Property

```
rgexp.global
```

Arguments: JScript Regular Expression Object global Property

rgexp

A **Regular Expression** object.

Remarks: JScript Regular Expression Object global Property

The **global** property contains **True** if the global (g) switch is used with a regular expression, and **False** if not.

JScript Regular Expression Object ignoreCase Property

Indicates whether the ignore case (i) switch has been used. Read-only.

Syntax: JScript Regular Expression Object ignoreCase Property

```
rgexp.ignoreCase
```

Arguments: JScript Regular Expression Object ignoreCase Property

rgexp

A **Regular Expression** object.

Remarks: JScript Regular Expression Object ignoreCase Property

The **ignoreCase** property contains **True** if the ignore case (i) switch is used with a regular expression, and **False** if not.

JScript Regular Expression Object lastIndex Property

Specifies the index at which to start the next match.

Syntax: JScript Regular Expression Object lastIndex Property

```
rgexp.lastIndex [= index]
```

Arguments: JScript Regular Expression Object lastIndex Property

rgexp

A **Regular Expression** object. Can be a variable name or a literal. Required.

index

The index from which to begin the next search.

Remarks: JScript Regular Expression Object lastIndex Property

The **lastIndex** property is modified by the **exec** method, and the **match**, **replace**, and **split** methods of the **String** object.

The following rules apply to values of **lastIndex**:

- If **lastIndex** is greater than the length of the string, the **test** and **exec** methods fail, and **lastIndex** is set to zero.
- If **lastIndex** is equal to the length of the string, the regular expression matches if the pattern matches the empty string. Otherwise, the match fails and **lastIndex** is reset to zero.
- Otherwise, **lastIndex** is set to the next position following the most recent match.

JScript Regular Expression Object source Property

Contains the text of the regular expression pattern. Read-only.

Syntax: JScript Regular Expression Object source Property

```
rgexp.source
```

Arguments: JScript Regular Expression Object source Property

rgexp

A **Regular Expression** object. It can be a variable name or a literal.

JScript Regular Expression Object test Method

Tests whether a pattern exists in a string.

```
rgexp.test(str)
```

Arguments: JScript Regular Expression Object test Method

rgexp

A **Regular Expression** object. Can be a variable name or a literal. Required.

str

The string to test a search on. Required.

Remarks: JScript Regular Expression Object test Method

The **test** method checks to see if a pattern exists within a string and returns **True** if so, and **False** otherwise.

The **RegExp** object is not modified by the **test** method.

JScript String Object

JScript String Object

The **String** object enables manipulation and formatting of text strings and determination and location of substrings within strings.

Methods: JScript String Object

JScript String Object anchor Method	Places an HTML anchor tag (<A>) with a NAME attribute around text.
JScript String Object big Method	Places HTML <BIG> tags around text.
JScript String Object blink Method	Places HTML <BLINK> tags around text.
JScript String Object bold Method	Places HTML <B> tags around text.
JScript String Object charAt Method	Retrieves the character as the specified index.
JScript String Object charCodeAt Method	Returns the Unicode encoding of the specified character.
JScript String Object concat Method	Combines two strings into one **String** object.
JScript String Object fixed Method	Places HTML <TT> tags around text.
JScript String Object fontcolor Method	Places an HTML <FONT> tag with the COLOR attribute around text.
JScript String Object fontsize Method	Places an HTML <FONT> tag with the SIZE attribute around text.
JScript String Object fromCharCode Method	Creates a string from a number of Unicode character values.
JScript String Object indexOf Method	Finds the first occurrence of a substring within a **String** object.
JScript String Object italics Method	Places HTML <I> tags around text.
JScript String Object lastIndexOf Method	Finds the last occurrence of a substring.
JScript String Object link Method	Places an HTML anchor tag (<A>) with an HREF attribute around text.
JScript String Object match Method	Performs a search on a string using the supplied **Regular Expression** object.
JScript String Object replace Method	Replaces the text found by a regular expression with other text.
JScript String Object search Method	Searches a string for matches to a regular expression.
JScript String Object slice Method	Returns a section of a string.

JScript String Object small Method	Places HTML <SMALL> tags around text.
JScript String Object split Method	Removes text from a string.
JScript String Object strike Method	Places HTML <STRIKE> tags around text.
JScript String Object sub Method	Places HTML <SUB> tags around text.
JScript String Object substr Method	Returns a substring beginning at a specified location and having a specified length.
JScript String Object substring Method	Returns the substring at a specified location.
JScript String Object sup Method	Places HTML <SUP> tags around text.
JScript String Object toLowerCase Method	Converts text to lowercase letters.
JScript String Object toUpperCase Method	Converts text to uppercase letters.

Properties: JScript String Object

JScript String Object length Property	The length of a **String** object.

Syntax: JScript String Object

```
StringObj[.method]

"String Literal"[.method]
```

Arguments: JScript String Object

method

A **String** object method.

Remarks: JScript String Object

String objects can be created implicitly using string literals. **String** objects created in this fashion (referred to as standard strings) are treated differently than **String** objects created using the **new** operator. All string literals share a common, global string object. So, if a property is added to a string literal, it is available to all standard string objects:

```
var alpha, beta;

alpha = "This is a string";

beta = "This is also a string";

alpha.test = 10;
```

In this example, *test* is now defined for *beta* and all future string literals. In the following example, however, added properties are treated differently:

```
var gamma, delta;

gamma = new String("This is a string");

delta = new String("This is also a string");

gamma.test = 10;
```

In this case, *test* is not defined for *delta*. Each **String** object declared as a **new String** object has its own set of members. This is the only case where **String** objects and string literals are handled differently.

JScript String Object anchor Method

Places an HTML anchor <A> with a NAME attribute around specified text in the object.

Syntax: JScript String Object anchor Method

```
strVariable.anchor(anchorstring)
"String Literal".anchor(anchorstring)
```

Arguments: JScript String Object anchor Method

strVariable

The name of a **String** object.

anchorstring

Text you want to place in the NAME attribute of an HTML anchor.

Remarks: JScript String Object anchor Method

Call the **anchor** method to create a named anchor out of a **String** object. The following example demonstrates how the **anchor** method accomplishes this:

```
var strVariable = "This is an anchor" ;
strVariable = strVariable.anchor("Anchor1");
```

The value of *strVariable* after the last statement is:

```
<A NAME="Anchor1">This is an anchor</A>
```

No checking is done to see if the tag already exists.

JScript String Object big Method

Places HTML <BIG> tags around text in a **String** object.

Syntax: JScript String Object anchor Method

```
strVariable.big( )
"String Literal".big( )
```

Arguments: JScript String Object anchor Method

strVariable

The name of a **String** object.

Remarks: JScript String Object anchor Method

The example that follows shows how the **big** method works:

```
var strVariable = "This is a string object";
strVariable = strVariable.big( );
```

The value of *strVariable* after the last statement is:

```
<BIG>This is a string object</BIG>
```

No checking is done to see if the tag already exists.

JScript String Object blink Method

Places HTML <BLINK> tags around text in a **String** object.

Syntax: JScript String Object blink Method

```
strVariable.blink( )
"String Literal".blink( )
```

Arguments: JScript String Object blink Method

strVariable

The name of a **String** object.

Remarks: JScript String Object blink Method

The example that follows demonstrates how the **blink** method works:

```
var strVariable = "This is a string object";
strVariable = strVariable.blink( );
```

The value of *strVariable* after the last statement is:

```
<BLINK>This is a string object</BLINK>
```

No checking is done to see if the tag already exists.

The <BLINK> tag is not supported in Microsoft Internet Explorer.

JScript String Object bold Method

Places HTML <B> tags around text in a **String** object.

Syntax: JScript String Object bold Method

```
strVariable.bold( )
"String Literal".bold( )
```

Arguments: JScript String Object bold Method

strVariable

The name of a **String** object.

Remarks: JScript String Object bold Method

The example that follows demonstrates how the **bold** method works:

```
var strVariable = "This is a string object";
strVariable = strVariable.bold( );
```

The value of *strVariable* after the last statement is:

```
<B>This is a string object</B>
```

No checking is done to see if the tag already exists.

JScript String Object charAt Method

Retrieves the character at the index specified.

Syntax: JScript String Object charAt Method

```
strVariable.charAt(index)
"String Literal".charAt(index)
```

Arguments: JScript String Object charAt Method

strVariable

The name of a **String** object.

index

The zero-based index of the desired character. Valid values are between 0 and the length of the string minus 1.

Remarks: JScript String Object charAt Method

The **charAt** method returns a character value equal to the character at the specified *index*. The first character in a string is at index 0, the second is at index 1, and so forth. Values of *index* out of valid range return undefined.

JScript String Object charCodeAt Method

Returns the Unicode encoding of the specified character.

Syntax: JScript String Object charCodeAt Method

```
stringObj.charCodeAt(index)
```

Arguments: JScript String Object charCodeAt Method

stringObj

The name of a **String** object.

index

The zero-based index of the specified character. Required.

Remarks: JScript String Object charCodeAt Method

If there is no character at the specified *index*, **NaN** is returned.

JScript String Object concat Method

Returns a **String** object containing the concatenation of two supplied strings.

Syntax: JScript String Object concat Method

```
string1.concat(string2)
```

Arguments: JScript String Object concat Method

string1

The **String** object or literal to concatenate with *string2*. Required.

string2

A **String** object or literal to concatenate to the end of *string1*. Required.

Remarks: JScript String Object concat Method

The result of the **concat** method is equivalent to: *result = string1 + string2.*

JScript String Object fixed Method

Places HTML <TT> tags around text in a **String** object.

Syntax: JScript String Object fixed Method

```
strVariable.fixed( )
"String Literal".fixed( )
```

Arguments: JScript String Object fixed Method

strVariable

A **String** object.

Remarks: JScript String Object fixed Method

The example that follows demonstrates how the **fixed** method works:

```
var strVariable = "This is a string object";
strVariable = strVariable.fixed( );
```

The value of *strVariable* after the last statement is:

```
<TT>This is a string object</TT>
```

No check is done to see if the tag already exists.

JScript String Object fontcolor Method

Places an HTML <FONT> tag with the COLOR attribute around the text in a **String** object.

Syntax: JScript String Object fontcolor Method

```
strVariable.fontcolor(colorval)

"String Literal".fontcolor(colorval)
```

Arguments: JScript String Object fontcolor Method

strVariable

A **String** object.

colorval

A string containing a color value. This can either be the hexadecimal value for a color, or the predefined name for a color.

Remarks: JScript String Object fontcolor Method

The following example demonstrates the **fontcolor** method:

```
var strVariable = "This is a string";

strVariable = strVariable.fontcolor("red");
```

The value of *strVariable* after the last statement is:

```
<FONT COLOR="RED">This is a string</FONT>
```

Valid predefined color names depend on your JScript host (browser, server, and so forth). They may also vary from version to version of your host. Check your host documentation for more information.

No checking is done to see if the tag has already been applied to the string.

JScript String Object fontsize Method

Places an HTML <FONT> tag with the SIZE attribute around the text in a **String** object.

Syntax: JScript String Object fontsize Method

```
strVariable.fontsize(intSize)

"String Literal".fontsize(intSize)
```

Arguments: JScript String Object fontsize Method

strVariable

A **String** object.

intSize

An integer value that determines the size of the text.

Remarks: JScript String Object fontsize Method

The following example demonstrates the **fontsize** method:

```
var strVariable = "This is a string";

strVariable = strVariable.fontsize(-1);
```

The value of *strVariable* after the last statement is:

```
<FONT SIZE="-1">This is a string</FONT>
```

Valid integer values depend on your Microsoft JScript host. See your host documentation for more information.

No checking is done to see if the tag already exists

JScript String Object fromCharCode Method

Creates a string from a number of Unicode character values.

Syntax: JScript String Object fromCharCode Method

```
String.fromCharCode(code1, code2, ..., coden)
```

Arguments: JScript String Object fromCharCode Method

code1, *code2*, …*coden*

The series of Unicode character values to convert into a string.

Remarks: JScript String Object fromCharCode Method

A **String** object need not be created before calling **fromCharCode**.

In the following example, test contains the string "plain":

```
var test = String.fromCharCode(112, 108, 97, 105, 110);
```

JScript String Object indexOf Method

Finds the first occurrence of a substring within a **String** object.

Syntax: JScript String Object indexOf Method

```
strVariable.indexOf(substring, startindex)

"String Literal".indexOf(substring, startindex)
```

Arguments: JScript String Object indexOf Method

strVariable

The name of a **String** object.

substring

The substring to search for within the **String** object.

startindex

An optional integer value specifying the index to begin searching within the **String** object. If omitted, searching begins at the beginning of the string.

Remarks: JScript String Object indexOf Method

The **indexOf** method returns an integer value indicating the beginning of the substring within the **String** object. If the substring is not found, a -1 is returned.

If *startindex* is negative, *startindex* is treated as zero. If it is larger than the greatest character position index, it is treated as the largest possible index.

Searching is performed from left to right. Otherwise, this method is identical to the **lastIndexOf** method.

JScript String Object italics Method

Places HTML <I> tags around text in a **String** object.

Syntax: JScript String Object italics Method

```
strVariable.italics( )

"String Literal".italics( )
```

Arguments: JScript String Object italics Method

strVariable

The name of a **String** object.

Remarks: JScript String Object italics Method

The example that follows demonstrates how the **italics** method works:

```
var strVariable = "This is a string";

strVariable = strVariable.italics( );
```

The value of *strVariable* after the last statement is:

```
<I>This is a string</I>
```

No check is done to see if the tag already exists.

JScript String Object lastIndexOf Method

Finds the last occurrence of a substring within a **String** object.

Syntax: JScript String Object lastIndexOf Method

```
strVariable.lastIndexOf(substring, startindex)

"String Literal".lastIndexOf(substring, startindex)
```

Arguments: JScript String Object lastIndexOf Method

strVariable

The name of a **String** object.

substring

The substring to search for within the **String** object.

startindex

An optional integer value specifying the index to begin searching within the **String** object. If omitted, searching begins at the end of the string.

Remarks: JScript String Object lastIndexOf Method

The **lastIndexOf** method returns an integer value indicating the beginning of the substring within the **String** object. If the substring is not found, a -1 is returned.

If *startindex* is negative, *startindex* is treated as zero. If it is larger than the greatest character position index, it is treated as the largest possible index.

Searching is performed right to left. Otherwise, this method is identical to the **indexOf** method.

JScript String Object length Property

Contains the length of a **String** object.

Syntax: JScript String Object length Property

```
strVariable.length

"String Literal".length
```

Remarks: JScript String Object length Property

The **length** property contains an integer that indicates the number of characters in the **String** object. The last character in the **String** object has an index of **length** - 1.

JScript String Object link Method

Places an HTML anchor <A> with an HREF attribute around the text in a **String** object.

Syntax: JScript String Object link Method

```
strVariable.link(linkstring)

"String Literal".link(linkstring)
```

Arguments: JScript String Object link Method

strVariable

The name of a **String** object.

linkstring

The text that you want to place in the HREF attribute of the HTML anchor.

Remarks: JScript String Object link Method

Call the **link** method to create a hyperlink out of a **String** object. The following is an example of how the method accomplishes this:

```
var strVariable = "This is a hyperlink";

strVariable = strVariable.link("http://www.microsoft.com");
```

The value of *strVariable* after the last statement is:

```
<A HREF="http://www.microsoft.com">This is a hyperlink</A>
```

No check is done to see if the tag already exists

JScript String Object match Method

Performs a search on a string using the supplied **Regular Expression** object.

Syntax: JScript String Object match Method

```
stringObj.match(rgExp)
```

Arguments: JScript String Object match Method

strObj

The name of a **String** object. Required.

rgExp

The **Regular Expression** object to use in the search. Required.

Remarks: JScript String Object match Method

The **match** method, which behaves like the **exec** method, returns an array of values.

Element zero of the array contains the last matched characters. Elements 1...*n* contain matches to any parenthesized substrings in the regular expression.

The method updates the contents of the **RegExp** object.

JScript String Object replace Method

Replaces the text found by a regular expression with other text.

Syntax: JScript String Object replace Method

```
stringObj.match(rgExp, replaceText)
```

Arguments: JScript String Object replace Method

stringObj

The name of a **String** object. Required.

rgExp

A **Regular Expression** object describing what to search for. Required.

replaceText

A **String** object or literal containing the text to replace for every successful match of *rgExp* in *stringObj*.

Remarks: JScript String Object replace Method

The result of the **replace** method is a copy of *stringObj* after all replacements have been made.

The method updates the contents of the **RegExp** object.

JScript String Object search Method

Searches a string for matches to a regular expression.

Syntax: JScript String Object search Method

```
stringObj.search(rgexp)
```

Arguments: JScript String Object search Method

stringObj

The name of a **String** object. Required.

rgExp

A **Regular Expression** object containing the pattern to search for. Required.

Remarks: JScript String Object search Method

The **search** method indicates if a match is present or not. It returns **True** if a match is found, and **False** otherwise. To get further information, use the **match** method.

JScript String Object slice Method

Returns a section of a string.

Syntax: JScript String Object slice Method

```
stringObj.slice(start, [end])
```

Arguments: JScript String Object slice Method

stringObj

A **String** object or literal. Required.

start

The zero-based index of the beginning of the specified portion of *stringObj*. Required.

end

The zero-based index of the end of the specified portion of *stringObj*. Optional.

Remarks: JScript String Object slice Method

The slice method returns a **String** object containing the specified portion of *stringObj*.

If negative, *end* indicates an offset from the end of *stringObj*. In addition, it is not zero-based. If omitted, extraction continues to the end of *stringObj*.

In the example that follows, the two uses of the **slice** method return the same thing. Negative one in the second example points to the last character in *str1* as the ending point:

```
str1.slice(0)

str2.slice(0,-1)
```

JScript String Object small Method

Places HTML <SMALL> tags around text in a **String** object.

Syntax: JScript String Object small Method

```
strVariable.small( )

"String Literal".small( )
```

Arguments: JScript String Object small Method

strVariable

The name of a **String** object.

Remarks: JScript String Object small Method

The example that follows demonstrates how the **small** method works:

```
var strVariable = "This is a string";

strVariable = strVariable.small( );
```

The value of *strVariable* after the last statement is:

```
<SMALL>This is a string</SMALL>
```

No checking is done to see if the tag already exists.

JScript String Object split Method

Removes text from a string.

Syntax: JScript String Object split Method

```
stringObj.split(rgExp)
```

Arguments: JScript String Object split Method

stringObj

The name of a **String** object. Required.

rgExp

A **Regular Expression** object containing the pattern to search for. Required.

Remarks: JScript String Object split Method

The result of the **split** method is a copy of *stringObj* after all successful matches have been removed.

The method updates the **RegExp** object.

JScript String Object strike Method

Places HTML <STRIKE> tags around text in a **String** object.

Syntax: JScript String Object strike Method

```
strVariable.strike( )

"String Literal".strike( )
```

Arguments: JScript String Object strike Method

strVariable

The name of a **String** object.

Remarks: JScript String Object strike Method

The example that follows demonstrates how the **strike** method works:

```
var strVariable = "This is a string object";

strVariable = strVariable.strike( );
```

The value of *strVariable* after the last statement is:

```
<STRIKE>This is a string object</STRIKE>
```

No check is done to see if the tag already exists.

JScript String Object sub Method

Places HTML <SUB> tags around text in a **String** object.

Syntax: JScript String Object sub Method

```
strVariable.sub( )

"String Literal".sub( )
```

Arguments: JScript String Object sub Method

strVariable

The name of a **String** object.

Remarks: JScript String Object sub Method

The example that follows demonstrates how the **sub** method works:

```
var strVariable = "This is a string object"

strVariable = strVariable.sub( );
```

The value of *strVariable* after the last statement is:

```
<SUB>This is a string object</SUB>
```

No checking is done to see if the tag already exists.

JScript String Object substr Method

Returns a substring beginning at a specified location and having a specified length.

Syntax: JScript String Object substr Method

```
stringvar.substr(start [, length ])
```

Arguments: JScript String Object substr Method

stringvar

A string literal or **String** object from which the substring is extracted. Required.

start

The starting position of the desired substring. The index of the first character in the string is zero. Required.

length

The number of characters to include in the returned substring. Optional.

Remarks: JScript String Object substr Method

If *length* is zero or negative, an empty string is returned. If not specified, the substring continues to the end of *stringvar*.

JScript String Object substring Method

Retrieves the substring at the specified location within a **String** object.

Syntax: JScript String Object substring Method

```
strVariable.substring(start, end)

"String Literal".substring(start, end)
```

Arguments: JScript String Object substring Method

strVariable

The name of a **String** object.

start

The zero-based index indicating the beginning of the substring.

end

The zero-based index indicating the end of the substring.

Remarks: JScript String Object substring Method

The **substring** method returns a **String** object containing the substring derived from the original object.

The **substring** method uses the lower of *start* and *end* as the beginning point of the substring. For example, strvar**.substring(**0, 3**)** and strvar**.substring(**3, 0**)** return the same substring.

The only exception to this is for negative parameters. If the first parameter is less than zero, it is treated as zero. If the second parameter is negative, it is set to the value of the first parameter.

The length of the substring is equal to the absolute value of the difference between *start* and *end*. For example, the length of the substring returned in strvar**.substring(**0, 3**)** and strvar**.substring(**3, 0**)** is three.

Finally, *start* and *end* can be strings. If so, these strings are coerced into integers if possible. If not, the value of the parameter is treated as zero.

JScript String Object sup Method

Places HTML <SUP> tags around text in a **String** object.

Syntax: JScript String Object sup Method

```
strVariable.sup( )

"String Literal".sup( )
```

Arguments: JScript String Object sup Method

strVariable

The name of a **String** object.

Remarks: JScript String Object sup Method

The example that follows demonstrates how the **sup** method works:

```
var strVariable = "This is a string object";

strVariable = strVariable.sup( );
```

The value of *strVariable* after the last statement is:

^{This is a string object}

No check is done to see if the tag already exists.

JScript String Object toLowerCase Method

Places the text in a **String** object in lowercase characters.

Syntax: JScript String Object toLowerCase Method

```
strVariable.toLowerCase( )
"String Literal".toLowerCase( )
```

Arguments: JScript String Object toLowerCase Method

strVariable

The name of a **String** object.

Remarks: JScript String Object toLowerCase Method

The **toLowerCase** method has no effect on nonalphabetic characters.

The following example demonstrates the effects of the **toLowerCase** method:

```
var strVariable = "This is a STRING object";
strVariable = strVariable.toLowerCase( );
```

The value of *strVariable* after the last statement is:

```
this is a string object
```

JScript String Object toUpperCase Method

Places the text in a **String** object in uppercase characters.

Syntax: JScript String Object toUpperCase Method

```
strVariable.toUpperCase( )
"String Literal".toUpperCase( )
```

Arguments: JScript String Object toUpperCase Method

strVariable

The name of a **String** object.

Remarks: JScript String Object toUpperCase Method

The **toUpperCase** method has no effect on nonalphabetic characters.

The following example demonstrates the effects of the **toUpperCase** method:

```
var strVariable = "This is a STRING object";
strVariable = strVariable.toUpperCase( );
```

The value of *strVariable* after the last statement is:

```
THIS IS A STRING OBJECT
```

JScript TextStream Object

JScript TextStream Object

The **TextStream** object facilitates sequential access to file.

Methods: JScript TextStream Object

JScript TextStream Object TextStream Close Method	Closes an open **TextStream** file.
JScript TextStream Object TextStream Read Method	Reads a specified number of characters.
JScript TextStream Object ReadAll Method	Reads an entire **TextStream**.
JScript TextStream Object ReadLine Method	Reads an entire line.
JScript TextStream Object Skip Method	Skips a specified number of characters.
JScript TextStream Object SkipLine Method	Skips the next line.
JScript TextStream Object Write Method	Writes a specified string.
JScript TextStream Object WriteBlankLines Method	Writes a specified number of newline characters.
JScript TextStream Object WriteLine Method	Writes a specified string and newline character.

Properties: JScript TextStream Object

JScript TextStream Object AtEndOfLine Property	**True** if the file pointer is before the end-of-line marker.
JScript TextStream Object AtEndOfStream Property	**True** if the file pointer is at the end of a file.
JScript TextStream Object Column Property	The column number of the current character position.
JScript TextStream Object Line Property	The current line number.

Syntax: JScript TextStream Object

```
TextStream.{property | method( )}
```

Arguments: JScript TextStream Object

The *property* and *method* arguments can be any of the properties and methods associated with the **TextStream** object.

Note

In actual usage, **TextStream** is replaced by a variable placeholder representing the **TextStream** object returned from the **FileSystemObject**.

Remarks: JScript TextStream Object

In the following code, `a` is the **TextStream** object returned by the **CreateTextFile** method on the **FileSystemObject**:

```
var fs = new ActiveXObject("Scripting.FileSystemObject")
var a = fs.CreateTextFile("c:\\testfile.txt", true)
a.WriteLine("This is a test.")
a.Close
```

WriteLine and **Close** are two methods of the **TextStream** object.

JScript TextStream Object AtEndOfLine Property

Returns **True** if the file pointer is positioned immediately before the end-of-line marker in a **TextStream** file, **False** if it is not. Read-only.

Syntax: JScript TextStream Object AtEndOfLine Property

object.**AtEndOfLine**

Arguments: JScript TextStream Object AtEndOfLine Property

object

The name of a **TextStream** object.

Remarks: JScript TextStream Object AtEndOfLine Property

The **AtEndOfLine** property applies only to **TextStream** files that are open for reading; otherwise, an error occurs.

The following code illustrates the use of the **AtEndOfLine** property:

```
function GetALine(filespec)
{
var fs, a, s, ForReading;
ForReading = 1, s = "";
fs = new ActiveXObject("Scripting.FileSystemObject");
a = fs.OpenTextFile(filespec, ForReading, false);
while (!a.AtEndOfLine)
{
s += a.Read(1);
}
a.Close( );
return s;
}
```

JScript TextStream Object AtEndOfStream Property

Returns **true** if the file pointer is at the end of a **TextStream** file; **false** if it is not. Read-only.

Syntax: JScript TextStream Object AtEndOfStream Property

```
object.AtEndOfStream
```

Arguments: JScript TextStream Object AtEndOfStream Property

object

The name of a **TextStream** object.

Remarks: JScript TextStream Object AtEndOfStream Property

The **AtEndOfStream** property applies only to **TextStream** files that are open for reading, otherwise, an error occurs.

The following code illustrates the use of the **AtEndOfStream** property:

```
function GetALine(filespec)
{
var fs, f, s, ForReading;
ForReading = 1, s = "";
fs = new ActiveXObject("Scripting.FileSystemObject");
f = fs.OpenTextFile(filespec, ForReading, false);
while (!f.AtEndOfStream)
s += f.ReadLine( );
f.Close( );
return s;
}
```

JScript TextStream Object TextStream Close Method

Closes an open **TextStream** file.

Syntax: JScript TextStream Object TextStream Close Method

```
object.Close( );
```

Arguments: JScript TextStream Object TextStream Close Method

object

The name of a **TextStream** object.

JScript TextStream Object Column Property

Read-only property that returns the column number of the current character position in a **TextStream** file.

Syntax: JScript TextStream Object Column Property

```
object.Column
```

Arguments: JScript TextStream Object Column Property

object

The name of a **TextStream** object.

Remarks: JScript TextStream Object Column Property

After a newline character has been written, but before any other character is written, **Column** is equal to 1.

JScript TextStream Object Line Property

Read-only property that returns the current line number in a **TextStream** file.

Syntax: JScript TextStream Object Line Property

```
object.Line
```

Arguments: JScript TextStream Object Line Property

object

The name of a **TextStream** object.

Remarks: JScript TextStream Object Line Property

After a file is initially opened and before anything is written, **Line** is equal to 1.

JScript TextStream Object TextStream Read Method

Reads a specified number of characters from a **TextStream** file and returns the resulting string.

Syntax: JScript TextStream Object TextStream Read Method

```
object.Read(characters)
```

Arguments: JScript TextStream Object TextStream Read Method

object

The name of a **TextStream** object. Required.

characters

The number of characters you want to read from the file. Required.

JScript TextStream Object ReadAll Method

Reads an entire **TextStream** file and returns the resulting string.

Syntax: JScript TextStream Object ReadAll Method

```
object.ReadAll( );
```

Arguments: JScript TextStream Object ReadAll Method

object

The name of a **TextStream** object.

JScript TextStream Object ReadLine Method

Reads an entire line (up to, but not including, the newline character) from a **TextStream** file and returns the resulting string.

Syntax: JScript TextStream Object ReadLine Method

```
object.ReadLine( )
```

Arguments: JScript TextStream Object ReadLine Method

object

The name of a **TextStream** object.

JScript TextStream Object Skip Method

Skips a specified number of characters when reading a **TextStream** file.

Syntax: JScript TextStream Object Skip Method

```
object.Skip(characters)
```

Arguments: JScript TextStream Object Skip Method

object

The name of a **TextStream** object. Required.

characters

Number of characters to skip when reading a file. Required.

Remarks: JScript TextStream Object Skip Method

Skipped characters are discarded.

JScript TextStream Object SkipLine Method

Skips the next line when reading a **TextStream** file.

Syntax: JScript TextStream Object SkipLine Method

```
object.SkipLine( )
```

Arguments: JScript TextStream Object SkipLine Method

object

The name of a **TextStream** object.

JScript TextStream Object Write Method

Writes a specified string to a **TextStream** file.

Syntax: JScript TextStream Object Write Method

```
object.Write(string)
```

Arguments: JScript TextStream Object Write Method

object

The name of a **TextStream** object. Required.

string

The text you want to write to the file. Required.

Remarks: JScript TextStream Object Write Method

Specified strings are written to the file with no intervening spaces or characters between each string. Use the **WriteLine** method to write a newline character or a string that ends with a newline character.

JScript TextStream Object WriteBlankLines Method

Writes a specified number of newline characters to a **TextStream** file.

Syntax: JScript TextStream Object WriteBlankLines Method

```
object.WriteBlankLines(lines)
```

Arguments: JScript TextStream Object WriteBlankLines Method

object

The name of a **TextStream** object. Required.

lines

The number of newline characters you want to write to the file. Required.

Remarks: JScript TextStream Object WriteBlankLines Method

For Windows systems, **WriteBlankLines** uses <CR><LF> as the newline character. On UNIX systems, **WriteBlankLines** uses <LF>.

JScript TextStream Object WriteLine Method

Writes a specified string and newline character to a **TextStream** file.

Syntax: JScript TextStream Object WriteLine Method

```
object.WriteLine([string])
```

Arguments: JScript TextStream Object WriteLine Method

object

The name of a **TextStream** object. Required.

string

The text you want to write to the file. If omitted, a newline character is written to the file. Optional.

Remarks: JScript TextStream Object WriteLine Method

For Windows systems, **WriteBlankLines** uses <CR><LF> as the newline character. On UNIX systems, **WriteBlankLines** uses <LF>.

JScript VBArray Object

JScript VBArray Object

The **VBArray** object provides access to Visual Basic safe arrays.

Methods: JScript VBArray Object

JScript VBArray Object Dimensions Method	The number of dimensions in a VBArray.
JScript VBArray Object getItem Method	Retrieves an item from a VBArray.
JScript VBArray Object lbound Method	The lowest index value used in a specified dimension.
JScript VBArray Object toArray Method	Converts the VBArray to a standard JScript array.
JScript VBArray Object ubound Method	The highest index value used in a specified dimension.

Syntax: JScript VBArray Object

```
new VBArray(safeArray)
```

Arguments: JScript VBArray Object

safeArray

A **VBArray** value.

Remarks: JScript VBArray Object

VBArray objects are read-only, and cannot be created directly. The *safeArray* argument must have obtained a **VBArray** value before being passed to the **VBArray** constructor. This can only be done by retrieving the value from an existing ActiveX or other object.

VBArrays can have multiple dimensions. The indices of each dimension can be different. The **dimensions** method retrieves the number of dimensions in the array; the **lbound** and **ubound** methods retrieve the range of indices used by each dimension.

JScript VBArray Object Dimensions Method

Returns the number of dimensions in a **VBArray** object.

Syntax: JScript VBArray Object Dimensions Method

```
array.dimensions( )
```

Arguments: JScript VBArray Object Dimensions Method

array

A **VBArray** object.

Remarks: JScript VBArray Object Dimensions Method

The **dimensions** method provides a way to retrieve the number of dimensions in a specified **VBArray** object.

JScript VBArray Object getItem Method

Retrieves the item at the specified location.

Syntax: JScript VBArray Object getItem Method

```
safeArray.getItem(dimension1[, dimension2, ...], dimensionN)
```

Arguments: JScript VBArray Object getItem Method

safeArray

A **VBArray** object. Required.

dimension1, ..., *dimensionN*

Specifies the exact location of the desired element of the **VBArray**. *N* equals the number of dimensions in the **VBArray**.

JScript VBArray Object lbound Method

Returns the lowest index value used in the specified dimension of a **VBArray**.

Syntax: JScript VBArray Object lbound Method

```
safeArray.lbound(dimension)
```

Arguments: JScript VBArray Object lbound Method

safeArray

A **VBArray** object. Required.

dimension

The dimension of the **VBArray** for which the lower bound index is wanted. If omitted, the **lbound** method behaves as if a 1 was passed. Optional.

Remarks: JScript VBArray Object lbound Method

If the **VBArray** is empty, the **lbound** method returns undefined. If *dimension* is greater than the number of dimensions in the **VBArray**, or is negative, the method generates a "Subscript out of range" error.

JScript VBArray Object toArray Method

Converts a **VBArray** to a standard JScript array.

Syntax: JScript VBArray Object toArray Method

```
safeArray.toArray( )
```

Arguments: JScript VBArray Object toArray Method

safeArray

A **VBArray** object.

Remarks: JScript VBArray Object toArray Method

The conversion translates the multidimensional **VBArray** object into a single dimensional JScript array. Each successive dimension is appended to the end of the previous one. For example, a **VBArray** object with three dimensions and three elements in each dimension is converted into a JScript array as follows

Suppose the **VBArray** object contains: (1, 2, 3), (4, 5, 6), (7, 8, 9). After translation, the JScript array contains: 1, 2, 3, 4, 5, 6, 7, 8, 9.

There is currently no way to convert a JScript array into a **VBArray** object.

JScript VBArray Object ubound Method

Returns the highest index value used in the specified dimension of the VBArray.

Syntax: JScript VBArray Object ubound Method

```
safeArray.ubound(dimension)
```

Arguments: JScript VBArray Object ubound Method

safeArray

A **VBArray** object. Required.

dimension

The dimension of the **VBArray** object for which the higher bound index is wanted. If omitted, **ubound** behaves as if a 1 was passed. Optional.

Remarks: JScript VBArray Object ubound Method

If the **VBArray** object is empty, the **ubound** method returns undefined. If *dimension* is greater than the number of dimensions in the **VBArray** object, or is negative, the method generates a "Subscript out of range" error.

JScript FileSystemObject Collections

Collections: JScript Collections

JScript Collections Drives Collection	Collection of available drives.
JScript Collections Files Collection	Collection of all **File** objects in a **Folder** object.
JScript Folders Collection	Collection of all **Folder** objects in a **Folder** object.

Methods: JScript Collections

JScript Collections Add Method	Adds a new **Folder** object to a **Folders** collection.

Properties: JScript Collections

JScript Collections Count Property	The number of items in a collection.
JScript Collections Item Property	An item in a collection.

Note

Collections returned by **FileSystemObject** method calls reflect the state of the file system when the collection was created. Changes to the file system after creation are not reflected in the collection. If the file system might be changed during the lifetime of the collection object, the method returning the collection should be called again to ensure that the contents are current.

JScript Collections Drives Collection

Syntax: JScript Collections Drives Collection

```
object.Drives
```

Arguments: JScript Collections Drives Collection

object

A **FileSystemObject**.

Remarks: JScript Collections Drives Collection

Removable-media drives need not have media inserted for them to appear in the **Drives** collection.

You can iterate the members of the **Drives** collection using the **Enumerator** object and the **for** statement:

```
function ShowDriveList()
{
var fs, s, n, e, x;
fs = new ActiveXObject("Scripting.FileSystemObject");
e = new Enumerator(fs.Drives);
s = "";
for (; !e.atEnd(); e.moveNext())
{
x = e.item();
s = s + x.DriveLetter;
s += " - ";
if (x.DriveType == 3)
n = x.ShareName;
else if (x.IsReady)
n = x.VolumeName;
else
n = "[Drive not ready]";
s += n + "<br>";
}
Response.Write(s);
}
```

Under UNIX the **Drives** collection has only one member, "/".

JScript Collections Files Collection

Collection of all **File** objects within a folder.

Remarks: JScript Collections Files Collection

The following code illustrates how to get a **Files** collection and iterate the collection using the **Enumerator** object and the **for** statement:

```
function ShowFolderFileList(folderspec)
{
var fs, f, f1, fc, s;
```

```
fs = new ActiveXObject("Scripting.FileSystemObject");
f = fs.GetFolder(folderspec);
fc = new Enumerator(f.files);
s = "";
for (; !fc.atEnd(); fc.moveNext())
{
s += fc.item();
s += "<br>";
}
Response.Write(s);
}
```

JScript Folders Collection

Collection of all **Folder** objects contained within a **Folder** object.

Remarks: JScript Folders Collection

The following code illustrates how to get a **Folders** collection and how to iterate the collection using the **Enumerator** object and the **for** statement:

```
function ShowFolderList(folderspec)
{
var fs, f, f1, fc, s;
fs = new ActiveXObject("Scripting.FileSystemObject");
f = fs.GetFolder(folderspec);
fc = new Enumerator(f.SubFolders);
s = "";
for (; !fc.atEnd(); fc.moveNext())
{
s += fc.item();
s += "<br>";
}
Response.Write(s);
}
```

JScript Folders Collection Add Method

Adds a new **Folder** to a **Folders** collection.

Syntax: JScript Folders Collection Add Method

```
object.Add (folderName)
```

Arguments: JScript Folders Collection Add Method

object

The name of a **Folders** collection. Required.

folderName

The name of the new **Folder** object being added. Required.

Remarks: JScript Folders Collection Add Method

The following example illustrates the use of the **Add** method to create a new folder:

```
function AddNewFolder(path, folderName)
{
var fs, f, fc, nf;
fs = new ActiveXObject("Scripting.FileSystemObject");
f = fs.GetFolder(path);
fc = f.SubFolders;
if (folderName != "" )
nf = fc.Add(folderName);
else
nf = fc.Add("New Folder");
}
```

An error occurs if the *folderName* already exists.

JScript Collections Count Property

Returns the number of items in a collection. Read-only.

Syntax: JScript Collections Count Property

```
object.Count
```

Arguments: JScript Collections Count Property

object

The name of one of the collections.

Remarks: JScript Collections Count Property

The following code illustrates use of the **Count** property:

```
var a, d, i, s; // Create some variables.
d = new ActiveXObject("Scripting.Dictionary");
d.Add ("a", "Athens"); // Add some keys and items
d.Add ("b", "Belgrade");
d.Add ("c", "Cairo");
a = (new VBArray(d.Keys())); // Get the keys.
s = "";
for (i = 0; i < d.Count; i++) //Iterate the dictionary.
{
s += a.getItem(i) + " - " + d(a.getItem(i)) + "<br>";
}
document.write(s); // Print item.
```

JScript Collections Item Property

Returns an *item* based on the specified *key*. Read/write.

Syntax: JScript Collections Item Property

```
object.Item(key)[ = newitem]
```

Arguments: JScript Collections Item Property

object

The name of a collection object. Required.

key

Index associated with the *item* being retrieved or added. Required.

Remarks: JScript Collections Item Property

If *key* is not found when attempting to return an existing item, a new *key* is created and its corresponding *item* is left empty.

SpicePack Component Reference

Sun SpicePack is a set of COM components that handle commonly used ASP application functionality. The components are Chili!Mail, Chili!POP3, and Chili!Upload. These components

can be instantiated and called from ASP scripts to send and receive e-mail and upload files from client browsers.

The components are installed with Sun Chili!Soft ASP and are enabled or disabled from the Sun Chili!Soft ASP Administration Console. This section provides information about enabling and disabling the SpicePack components, and reference information about using the components.

In this section:

- Enabling SpicePack Components
- Chili!Mail (SMTP) Component
- Chili!POP3 (POP3) Component
- Chili!Upload (File Upload) Component

Enabling SpicePack Components

SpicePack components are enabled or disabled from the Sun Chili!Soft ASP Administration Console, using the following procedure.

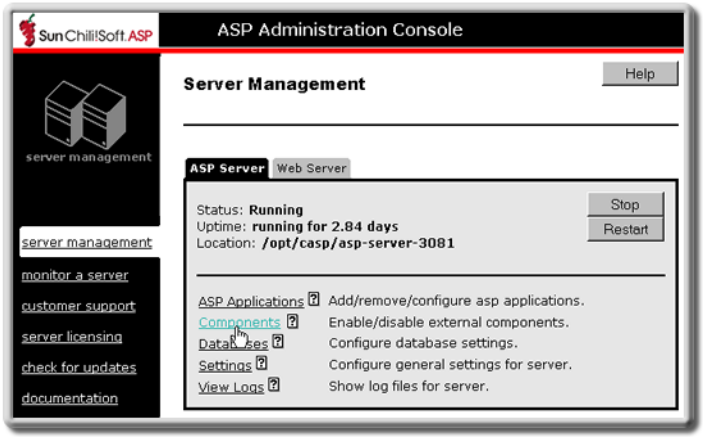
To enable or disable SpicePack components

1. If necessary, open the Administration Console by using the following URL:

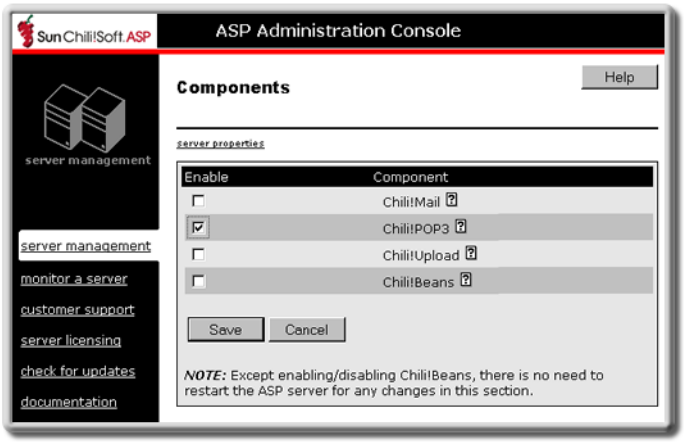
   http://[HOSTNAME]:[PORT}

   where [HOSTNAME] is the hostname of your Web server and [PORT] is the port on which the Administration Console is running (5100 by default).

2. On the **ASP Server** tab of the **Server Management** page (the first page to display when you open the Administration Console), click **Components**.



   The **Components** page displays.



3. Click to select or clear (enable or disable) the **Chili!Mail**, **Chili!POP3**, and **Chili!Upload** check boxes as desired.
4. Click **Save** to save your changes.

   - or -

   Click **Cancel** to revert to the last settings that were saved.

   Note: If you enabled or disabled the Chili!Mail, Chili!POP3, or Chili!Upload components, you do not need to restart the ASP Server.

See also:

SpicePack Component Reference in this chapter

Chili!Mail (SMTP) Component in this chapter

Chili!POP3 (POP3) Component in this chapter

Chili!Upload (File Upload) Component in this chapter

Chili!Mail (SMTP) Component

The Chili!Mail component enables users to send e-mail messages from an ASP page to an SMTP e-mail server. The Chili!Mail component is compatible with the **NewMail** object included with the Microsoft Internet Information Services (IIS) CDONTS component. However, the Chili!Mail component does not support the following properties and methods of the **NewMail** object:

- **AttachURL**
- **ContentBase**
- **ContentLocation**
- **MailFormat**
- **SetLocaleIDs**
- **Version**

Other differences between the Microsoft **NewMail** object and the Chili!Mail component are described in the property and method descriptions that follow.

Chili!Mail Registry Settings

The Chili!Mail component does not use registry settings.

Chili!Mail Syntax

The Chili!Mail component is registered with the ProgId of "CDONTS.NewMail".

The following ASP script written in VBScript creates an instance of the component:

```
Set mailer = Server.CreateObject("CDONTS.NewMail")
```

Chili!Mail Properties

The Chili!Mail component exposes the following properties:

- **Bcc**
- **Body**
- **BodyFormat**
- **Cc**
- **From**
- **Host**
- **Importance**
- **Retain**
- **Subject**
- **To**
- **Value**
- **WrapLength**

Chili!Mail Bcc Property (String: Read/Write)

The **Bcc** property specifies one or more recipients of a blind copy of the message. A full messaging address must be provided for each recipient, as shown in the following example:

```
"useraddress@company.com"
```

Addresses must be separated by a semicolon (;), as shown in the following example:

```
"user1@company1.com;user2@company2.com;user3@company3.com"
```

Chili!Mail Body Property (String: Read/Write)

The **Body** property is a string that specifies the content of the message. Line breaks should be sent as carriage return-linefeed pairs, for example, "Chr(13) & Chr(10)".

Chili!Mail BodyFormat Property (Long: Write only)

The **BodyFormat** property specifies the message format available for the Chili!Mail **Body** property. The values for the **BodyFormat** property can be set as follows:

- 0 indicates that the **Body** property can include HTML
- 1 indicates that the **Body** property can include plain text only (default)

Chili!Mail Cc Property (String: Read/Write)

The **Cc** property specifies one or more recipients of a copy of the message. A full messaging address must be provided for each recipient, as shown in the following example:

```
"useraddress@company.com"
```

Addresses must be separated by a semicolon (;), as shown in the following example:

```
"user1@company1.com;user2@company2.com;user3@company3.com"
```

Chili!Mail From Property (String: Read/Write)

The **From** property is a string that specifies the content of the From field of the message header. It cannot include spaces.

Note

The From field cannot exceed 255 characters, the limit for a single e-mail address. There is no character limit for the **To**, **Cc**, and **Bcc** fields.

Chili!Mail Host Property (String: Read/Write)

The **Host** property is a string that specifies the valid DNS name (for example, "mail.myorg.com") or IP address of the SMTP mail server. The default is "localhost".

Chili!Mail Importance Property (Long: Read/Write)

The **Importance** property specifies the importance of the message to be sent. Valid values are:

- 0 indicates low importance
- 1 indicates normal importance
- 2 indicates high importance

Chili!Mail Retain Property (BOOLEAN: Read/Write)

The **Retain** property specifies whether message properties are retained after the **Send** method is called. If set to **True**, all properties are retained. If set to **False** (the default), all properties are cleared.

Chili!Mail Subject Property (String: Read/Write)

The **Subject** property is a string that specifies the content of the subject line of the message. This property may be left empty.

Chili!Mail To Property (String: Read/Write)

The **To** property specifies one or more message recipients. A full messaging address must be provided for each recipient, as shown in the following example:

```
"useraddress@company.com"
```

Addresses must be separated by a semicolon (;), as shown in the following example:

```
"user1@company1.com;user2@company2.com;user3@company3.com"
```

If both the **To** property and the **To** parameter of the **Send** method are supplied, the message is sent to all recipients in both lists.

Chili!Mail Value Property (Read/Write)

The **Value** property adds one or more headers to the automatically generated headers, such as To, From, Subject, and Date. Possibilities for additional headers include File, Keywords, and Reference.

Certain headers, such as Reply-To, are widely accepted and used by various messaging systems. For such a header to be recognized by recipients, the character string in the header name must exactly match the accepted string.

In principle, you can put any combination of ASCII characters in the string, but some messaging systems might restrict the character set. The safest procedure is to limit the string to alphanumeric characters, dashes, and slashes, and in particular to avoid spaces.

You can set the **Value** property more than once. Each setting generates another header to be included with the existing headers.

Chili!Mail WrapLength (Read/Write)

The **WrapLength** property applies to message content. It specifies the maximum number of characters allowed in a line before the line wraps; in other words, before it breaks and continues on the next line. The line breaks at the last space before the specified maximum number of characters has been reached. The default setting is 76. The maximum is 1,000.

Chili!Mail Methods

The Chili!Mail component provides the following methods:

- **AttachFile**
- **Send**

Chili!Mail AttachFile Method

The **AttachFile** method attaches a file to the message. Messages are multi-part mime encoded, and attachments follow the text portion of the message.

Source	A string containing the absolute path name of the file to attach.

CDONTS Note

All messages are Base64 encoded. There is no provision for specifying a different encoding method.

Chili!Mail Send Method

The **Send** method sends the message using the properties previously set. All arguments to this method are optional and override the properties previously set for the message (except for the **To** argument, which is combined with any previously set **To** property).

Calling the **Send** method resets all message properties in preparation for the next message, unless the **Retain** property is set to **True**. Multiple messages can be sent using the same instance of the Chili!Mail component.

Chili!Mail Send Method Arguments

From	See the description of the property of the same name above.
To	See the description of the property of the same name above.
Subject	See the description of the property of the same name above.
Body	See the description of the property of the same name above.
Importance	See the description of the property of the same name above.
Host	See the description of the property of the same name above.

Chili!Mail Send Method Examples

Example 1:

```
Set mailmsg = Server.CreateObject("CDONTS.NewMail")

mailmsg.To = "youraccount@yourco.com"

mailmsg.From = "MailTest"

mailmsg.Body = "This is a test message." & Chr(13) & Chr(10) & "This
is the second line."

mailmsg.Host = "mail.yourco.com"

mailmsg.Send
```

Example 2:

```
Set mailmsg = Server.CreateObject("CDONTS.NewMail")
```

```
Message = "This is a test message." & Chr(13) & Chr(10) & "This is
the second line."

mailmsg.Send "myaccount@yourco.com", "youraccount@yourco.com", "Test
Subject", Message, 2, "mail.yourco.com"
```

Chili!POP3 (POP3) Component

The Chili!POP3 component retrieves e-mail messages from a POP3 server from an ASP script. This component has two main interfaces. The POP3 interface creates and controls the connection to a POP3 server. The Message interface exposes all of the properties of a single message. Additional interfaces are exposed to support retrieval of message lists and message attachments.

Chili!POP3 Registry Settings

The Chili!POP3 component does not use registry settings.

Chili!POP3 Syntax

The Chili!POP3 component is registered with the ProgId of "CHILI.POP3.1."

The following ASP script written in VBScript creates an instance of the component:

```
Set pop3 = Server.CreateObject( "CHILI.POP3.1" )
```

Chili!POP3 POP3 Interface

The POP3 interface creates and controls a connection to a POP3 server.

POP3 Interface Properties

The POP3 interface exposes no properties.

POP3 Interface Collections

- **Messages**

POP3 Interface Messages Collection

The **Messages** collection is a collection of **Message** objects, as described later in "Chili!POP3 Message Interface." This collection is read-only and does not support the standard **Append** or **Delete** collection methods.

POP3 Interface Methods

The following methods control a network connection to a POP3 server.

- **Connect**
- **Delete**
- **Disconnect**

- **Reset**

POP3 Interface Connect Method

The **Connect** method establishes a network connection to a POP3 server.

Arguments:

Host	The hostname of the server with which to connect.
UserId	The User ID required for connecting with the server.
Password	The password required for connecting with the server.

Example:

See the following **Disconnect** example.

POP3 Interface Disconnect Method

The **Disconnect** method disconnects from the POP3 server.

Example:

```
Set pop3 = Server.CreateObject("CHILI.POP3.1")
pop3.Connect "mail.foo.com", "myuserid", "mypsswd"
pop3.Disconnect
```

POP3 Interface Delete Method

The **Delete** method deletes a message on the POP3 server. This does not delete the message from the **Messages** collection.

Arguments:

Id	0-based index for the message in the message collection.

Reset:

Returns the POP3 server to the beginning of the transaction state (Connected) and ignores any commands and their effect on the connection. For example, any messages that were deleted from the mailbox are restored to their un-deleted state.

Example:

```
Set pop3 = Server.CreateObject("CHILI.POP3.1")
pop3.Connect "mail.foo.com", "myuserid", "mypsswd"
pop3.Reset
pop3.Disconnect
```

Chili!POP3 Message Interface

The Chili!POP3 component **Message** interface provides access to the messages currently in the mail store on the connected server. The properties, methods, and collections of the **Message** object are used to access those messages.

There are varying network costs associated with accessing the different properties of a message. For POP3 servers that support the optional **TOP** command, accessing any header information and the first few lines of the message can be accomplished without paying the data transfer overhead of moving the entire message from the server to the client.

Note

When requesting any of the properties that can be gathered without retrieving the entire message, the component first attempts the **TOP** command. If that command fails, the component then attempts to fulfill the property request via the full message **RETR** command.

Message Interface Properties

(LW means it can be "lightweight" on POP3 servers supporting the **TOP** command.)

- **From(LW)**
- **Subject(LW)**
- **Size(LW)**
- **DateSent(LW)**
- **DateReceived(LW)**
- **MsgId(LW)**
- **MsgUID(LW)**
- **HasAttachments(LW)**
- **Message**

Message Interface From Property (Read Only)

The **From** property is a string that indicates who sent the e-mail message.

Message Interface Subject Property (Read Only)

The **Subject** property is a string that indicates the subject of the message. It may be **Null** (empty string).

Message Interface Size Property (Read Only)

The **Size** property indicates the total size of the current message in bytes.

Message Interface DateSent Property (Read Only)

The **DateSent** property indicates the date and time that the message was sent.

Message Interface DateReceived Property (Read Only)

The **DateReceived** property indicates the date and time that the message was received.

Message Interface MsgId Property (Read Only)

The **MsgId** property indicates the message ID of the current message in the collection.

Message Interface MsgUID Property (Read Only)

The **MsgUID** property indicates whether the server supports the **UIDL** command. It returns **0** if the server does not support this command.

Message Interface HasAttachments Property (Read Only)

The **HasAttachments** property provides an "educated guess" based on the message headers (to be lightweight) as to whether the message has attachments.

Message Interface Message Property (Read Only)

The **Message** property is a string that specifies the content of the message.

Message Interface Collections

The **Message** interface collections are as follows:

- **To(LW)**
- **CC(LW)**
- **Headers(LW)**
- **Attachments**

Message Interface To Collection

The **To** collection is the list of e-mail addresses to which the message was sent. The collection is read-only and does not support the standard **Append** or **Delete** methods.

Message Interface CC Collection

The **CC** collection is the list of e-mail addresses to which the message was sent as a carbon copy. The collection is read-only and does not support the standard **Append** or **Delete** methods.

Message Interface Attachments Collection

The **Attachments** collection is the list of attachments to the current e-mail message, consisting of file name(s) and description(s). The collection is read-only and does not support the standard **Append** or **Delete** methods.

Message Interface Headers Collection

The **Interface Headers** collection is a collection of all message headers for the current e-mail message. This includes headers that are also accessible via friendly named fields or other collections, such as **To**, **From**, and **DateSent**. This collection can be accessed via the header name or index. This collection is read-only and does not support the standard **Append** or **Delete** methods.

Message Interface Methods

- **PreviewMessage**
- **SaveAttachments**

Message Interface PreviewMessage Method

The **PreviewMessage** method returns the specified number of lines of the message body. For servers that support the **TOP** command, this is performed without retrieving the entire message body. For messages with attachments or messages that consist entirely of binary data (which may be ascertained via the **Headers** collection) the first N lines of the message might not be meaningful to a human reader.

Message Interface PreviewMessage Arguments

Lines	The number of lines to return.

Message Interface SaveAttachments Method

The **SaveAttachments** method saves e-mail attachments to a specified directory on the server.

Message Interface SaveAttachments Arguments

Directory path on the Server	The complete path name to the directory on the server where attachments are to be saved.

Note

In a shared Web hosting environment, such as an ISP, you might not know the directory structure above the document root for your virtual host. In this situation, you cannot specify an absolute path name for the file, so you must use the **Server.mapPath** directive instead.

Note about To, CC, and Headers

To, **CC**, **BCC**, and **Headers** are **BSTR** collections using the **Count** method to obtain the total number of items in the collection and the **Item** method to obtain each item. The difference is that for **To**, **CC**, and **BCC**, the first argument of **Item** is a 0-based index, while for **Headers**, the first argument is a string that indicates the name of the header item (for example, **From**, **To**, **Subject**, and so forth).

Message Interface SaveAttachments Example [0]

```
Set pop3 = Server.CreateObject("CHILI.POP3.1")

pop3.Connect "mail.foo.com", "myuserid", "mypsswd"

For each item in pop3.Messages

   For each CC in Item.CC

         MsgBox CC

   next
```

```
next
pop3.Reset
pop3.Disconnect
```

Chili!POP3 Attachment Interface

The Chili!POP3 **Attachments** collection of the **Message** object provides access to the attachments currently in an e-mail. The properties and methods of the **Attachment** object are used to access those attachments.

Attachment Interface Properties

- **FileName**
- **ContentType**
- **FileSize**
- **Base64**

Attachment Interface FileName Property (Read Only)

The **FileName** property is a string that indicates the name of the attachment.

Attachment Interface ContentType Property (Read Only)

The **ContentType** property is a string that indicates the content type of the attachment.

Attachment Interface FileSize Property (Read Only)

The **FileSize** property is a number that indicates the size of the attachment in bytes.

Attachment Interface Base64 Property (Read Only)

The **Base64** property is a Boolean value that indicates whether the attachment is Base64 encoded.

Attachment Interface Methods

- **SaveToFile**
- **Read**

Attachment Interface SaveToFile Method

The **SaveToFile** method saves the attachment on the server.

Attachment Interface SaveToFile Arguments

Directory	The full directory path on the server.

Note

In a shared Web hosting environment, such as with an Internet Service Provider, you might not know the directory structure above the document root for your virtual host. In this situation, you cannot specify an absolute path name for the file, so you must use the **Server.mapPath** directive instead.

Attachment Interface Read Method

The **Read** method reads the attachment.

Attachment Interface Read Arguments

Nsize	The number of bytes to read from the attachment. This argument is optional. If missing, the entire attachment is read.
Pbytes	A safe array of bytes.

Chili!Upload (File Upload) Component

The Chili!Upload component enables users to save files uploaded by site visitors to the server.

Chili!Upload Registry Settings

The component does not use registry settings.

Chili!Upload Syntax

The Chili!Upload component is registered with the ProgId of "Chili.Upload.1." The following VBScript excerpt creates an instance of the control.

```
Set report = Server.CreateObject( "Chili.Upload.1" )
```

Chili!Upload Properties

The Chili!Upload component exposes the following properties:

- **AllowOverwrite**
- **SizeLimit**
- **FileSize**
- **SourceFileName**
- **SourceFileExtension**

Chili!Upload AllowOverwrite Property (Read /Write)

The **AllowOverwrite** property determines whether the component overwrites existing files saved with the same absolute path name as the uploaded file.

Chili!Upload SizeLimit Property (Read/Write)

The **SizeLimit** property sets the maximum file size in bytes of uploaded files.

Chili!Upload FileSize Property (Read Only)

The **FileSize** property indicates the size in bytes of the uploaded file.

Chili!Upload SourceFileName Property (Read Only)

The **SourceFileName** property indicates the file name of the uploaded file.

Chili!Upload SourceFileExtension Property (Read Only)

The **SourceFileExtension** property indicates the file extension of the uploaded file.

Chili!Upload Methods

The Chili!Upload component exposes the following method:

- **SaveToFile**

Chili!Upload SaveToFile Method

The **SaveToFile** method saves the uploaded file to the location specified by the absolute path name provided by the user.

Chili!Upload SaveToFile Arguments

Path	The absolute path name for the file, which specifies where it is to be saved.

Example:

The following script uploads a file:

```
<FORM ACTION="fileupld.asp" METHOD="POST" ENCTYPE="multipart/form-
data">
<INPUT TYPE="FILE" NAME="FILE">
<INPUT TYPE="SUBMIT" VALUE="Send">
</FORM>
```

The following fileupld.asp script processes the upload:

```
<%
Response.Expires = 0
Set fbase = Server.CreateObject("Chili.Upload.1")
fbase.SizeLimit = 10000
fbase.SaveToFile("/opt/datafiles/test.dat")
%>
```

```
Done writing <%=fbase.FileSize%> bytes from user file
<%=fbase.SourceFileName%> (of type <%=fbase.SourceFileExtension%>)
```

Note

In a shared Web hosting environment, such as with an Internet Service Provider, you might not know the directory structure above the document root for your virtual host. In this situation, you cannot specify an absolute path name for the file, so you must use the **Server.mapPath** directive instead. The following example saves the uploaded file to the document root of the virtual host:

```
<%

Response.Expires = 0

Set fbase = Server.CreateObject("Chili.Upload.1")

fbase.SizeLimit = 10000

fbase.SaveToFile("Server.mapPath("/") & "/" & "test.dat")

%>

Done writing <%=fbase.FileSize%> bytes from user file
<%=fbase.SourceFileName%> (of type
<%=fbase.SourceFileExtension%>)
```

VBScript Language Reference

Sun Chili!Soft ASP supports version 3.1 of Microsoft Visual Basic Scripting Edition (VBScript).

This section provides reference information on the following VBScript topics:

- VBScript Constants
- VBScript Operators
- VBScript Statements
- VBScript Functions
- VBScript Objects and Collections

VBScript Constants

The following is a list of VBScript constants:

- Color
- Comparison
- Date Format

- Date/Time
- DriveType
- File Attribute
- File InputOutput
- Miscellaneous
- MsgBox
- SpecialFolder
- String
- Tristate
- VarType

VBScript Color Constants

VBScript Color Constants are only available when your project has an explicit reference to the appropriate type library containing these constant definitions. For VBScript, you must explicitly declare these constants in your code.

Constant	Value	Description
vbBlack	&h00	Black
vbRed	&hFF	Red
vbGreen	&hFF00	Green
vbYellow	&hFFFF	Yellow
vbBlue	&hFF0000	Blue
vbMagenta	&hFF00FF	Magenta
vbCyan	&hFFFF00	Cyan
vbWhite	&hFFFFFF	White

VBScript Comparison Constants

VBScript Comparison Constants are only available when your project has an explicit reference to the appropriate type library containing these constant definitions. For VBScript, you must explicitly declare these constants in your code.

Constant	Value	Description
vbBinaryCompare	0	Perform a binary comparison.
vbTextCompare	1	Perform a textual comparison.

VBScript Date/Time Constants

VBScript Date/Time Constants are only available when your project has an explicit reference to the appropriate type library containing these constant definitions. For VBScript, you must explicitly declare these constants in your code.

Constant	Value	Description
vbSunday	1	Sunday
vbMonday	2	Monday
vbTuesday	3	Tuesday
vbWednesday	4	Wednesday
vbThursday	5	Thursday
vbFriday	6	Friday
vbSaturday	7	Saturday
vbFirstJan1	1	Use the week in which January 1 occurs (default).
vbFirstFourDays	2	Use the first week that has at least four days in the new year.
vbFirstFullWeek	3	Use the first full week of the year.
vbUseSystem	0	Use the date format contained in the regional settings for your computer.
vbUseSystemDayOfWeek	0	Use the day of the week specified in your system settings for the first day of the week.

VBScript Date Format Constants

VBScript Date Format Constants are only available when your project has an explicit reference to the appropriate type library containing these constant definitions. For VBScript, you must explicitly declare these constants in your code.

Constant	Value	Description
vbGeneralDate	0	Display a date and/or time. For real numbers, display a data and time. If there is no fractional part, display only a date. If there is no integer part, display time only. Date and time display is determined by your system

		settings.
vbLongDate	1	Display a date using the long date format specified in your computer's regional settings.
vbShortDate	2	Display a date using the short date format specified in your computer's regional settings.
vbLongTime	3	Display a time using the long time format specified in your computer's regional settings.
vbShortTime	4	Display a time using the short time format specified in your computer's regional settings.

VBScript DriveType Constants

VBScript DriveType Constants are only available when your project has an explicit reference to the appropriate type library containing these constant definitions. For VBScript, you must explicitly declare these constants in your code.

Constant	Value	Description
Unknown	0	Drive type can't be determined.
Removable	1	Drive has removable media. This includes all floppy drives and many other varieties of storage devices.
Fixed	2	Drive has fixed (nonremovable) media. This includes all hard drives, including hard drives that are removable.
Remote	3	Network drives. This includes drives shared anywhere on a network.
CDROM	4	Drive is a CD-ROM. No distinction is made between read-only and read/write CD-ROM drives.
RAMDisk	5	Drive is a block of Random Access Memory (RAM) on the local computer that behaves like a disk drive.

VBScript File Attribute Constants

VBScript File Attribute Constants are only available when your project has an explicit reference to the appropriate type library containing these constant definitions. For VBScript, you must explicitly declare these constants in your code.

Note

The applicability of these constants depends on the underlying operating system. If the OS file system does not support the file attribute requested, an error will be returned.

Constant	Value	Description
Normal	0	Normal file. No attributes are set.
ReadOnly	1	Read-only file.
Hidden	2	Hidden file.
System	4	System file.
Volume	8	Disk drive volume label.
Directory	16	Folder or directory.
Archive	32	File has changed since last backup.
Alias	64	Link or shortcut.
Compressed	128	Compressed file.

VBScript File InputOutput Constants

VBScript File InputOutput Constants are only available when your project has an explicit reference to the appropriate type library containing these constant definitions. For VBScript, you must explicitly declare these constants in your code.

Constant	Value	Description
ForReading	1	Open a file for reading only. You can't write to this file.
ForWriting	2	Open a file for writing. If a file with the same name exists, its previous contents are overwritten.
ForAppending	8	Open a file and write to the end of the file.

VBScript Miscellaneous Constants

VBScript Miscellaneous Constants are only available when your project has an explicit reference to the appropriate type library containing these constant definitions. For VBScript, you must explicitly declare these constants in your code.

Constant	Value	Description
vbObjectError	-2147221504	User-defined error numbers should be greater

than this value, for example,

```
Err.Raise Number = vbObjectError
+ 1000
```

VBScript MsgBox Constants

The following VBScript MsgBox Constants are used with the **MsgBox** function to identify what buttons and icons appear on a message box and which button is the default. In addition, the modality of the **MsgBox** can be specified. These constants are only available when your project has an explicit reference to the appropriate type library containing these constant definitions. For VBScript, you must explicitly declare these constants in your code.

Constant	Value	Description
vbOKOnly	0	Display OK button only.
vbOKCancel	1	Display OK and Cancel buttons.
vbAbortRetryIgnore	2	Display Abort, Retry, and Ignore buttons.
vbYesNoCancel	3	Display Yes, No, and Cancel buttons.
vbYesNo	4	Display Yes and No buttons.
vbRetryCancel	5	Display Retry and Cancel buttons.
vbCritical	16	Display Critical Message icon.
vbQuestion	32	Display Warning Query icon.
vbExclamation	48	Display Warning Message icon.
vbInformation	64	Display Information Message icon.
vbDefaultButton1	0	First button is the default.
vbDefaultButton2	256	Second button is the default.
vbDefaultButton3	512	Third button is the default.
vbDefaultButton4	768	Fourth button is the default.
vbApplicationModal	0	Application modal. The user must respond to the message box before continuing work in the current application.
vbSystemModal	4096	System modal. All applications are suspended until the user responds to the message box.

The following VBScript MsgBox Constants are used with the **MsgBox** function to identify which button a user has selected. These constants are only available when your project has an explicit

reference to the appropriate type library containing these constant definitions. For VBScript, you must explicitly declare these constants in your code.

Constant	Value	Description
vbOK	1	**OK** button was clicked.
vbCancel	2	**Cancel** button was clicked.
vbAbort	3	**Abort** button was clicked.
vbRetry	4	**Retry** button was clicked.
vbIgnore	5	**Ignore** button was clicked.
vbYes	6	**Yes** button was clicked.
vbNo	7	**No** button was clicked.

VBScript SpecialFolder Constants

VBScript SpecialFolder Constants are only available when your project has an explicit reference to the appropriate type library containing these constant definitions. For VBScript, you must explicitly declare these constants in your code.

Constant	Value	Description
WindowsFolder	0	The Windows folder contains files installed by the Windows operating system.
SystemFolder	1	The System folder contains libraries, fonts, and device drivers.
TemporaryFolder	2	The Temp folder is used to store temporary files. Its path is found in the TMP environment variable.

VBScript String Constants

The following VBScript String Constants can be used anywhere in your code in place of actual values:

Constant	Value	Description
vbCr	Chr(13)	Carriage return
vbCrLf	Chr(13) & Chr(10)	Carriage return-linefeed combination
vbFormFeed	Chr(12)	Form feed; not useful in Microsoft

		Windows
vbLf	Chr(10)	Line feed
vbNewLine	Chr(13) & Chr(10) or Chr(10)	Platform-specific newline character; whatever is appropriate for the platform
vbNullChar	Chr(0)	Character having the value 0
vbNullString	String having value 0	Not the same as a zero-length string (""); used for calling external procedures
vbTab	Chr(9)	Horizontal tab
vbVerticalTab	Chr(11)	Vertical tab; not useful in Microsoft Windows

VBScript Tristate Constants

VBScript Tristate Constants are only available when your project has an explicit reference to the appropriate type library containing these constant definitions. For VBScript, you must explicitly declare these constants in your code.

Constant	Value	Description
TristateTrue	-1	True
TristateFalse	0	False
TristateUseDefault	-2	Use default setting

VBScript VarType Constants

VBScript VarType Constants are only available when your project has an explicit reference to the appropriate type library containing these constant definitions. For VBScript, you must explicitly declare these constants in your code.

Constant	Value	Description
vbEmpty	0	Uninitialized (default)
vbNull	1	Contains no valid data
vbInteger	2	Integer subtype
vbLong	3	Long subtype
vbSingle	4	Single subtype

vbDouble	5	Double subtype
vbCurrency	6	Currency subtype
vbDate	7	Date subtype
vbString	8	String subtype
vbObject	9	Object
vbError	10	Error subtype
vbBoolean	11	Boolean subtype
vbVariant	12	Variant (used only for arrays of variants)
vbDataObject	13	Data access object
vbDecimal	14	Decimal subtype
vbByte	17	Byte subtype
vbArray	8192	Array

VBScript Operators

Operator	Description
VBScript Addition Operator (+)	Sum two numbers.
VBScript And Operator	Perform a logical conjunction on two expressions.
VBScript Assignment Operator (=) [VBScript Assignment Operator ('equals')]	Assign a value to a variable or property.
VBScript Concatenation Operator (&)	Force string concatenation of two expressions.
VBScript Division Operator (/)	Divide two numbers and return a floating point result.
VBScript Eqv Operator	Perform a logical equivalence on two expressions.
VBScript Exponentiation Operator (^)	Raise a number to the power of an exponent.
VBScript Imp Operator	Perform a logical implication on two expressions.
VBScript Integer Division Operator (\)	Divide two numbers and return an integer result.

VBScript Is Operator	Compare two object reference values.
VBScript Mod Operator	Divide two numbers and return the remainder.
VBScript Muliplication Operator (*)	Multiply two numbers.
VBScript Negation and Subtraction Operator (-)	Indicate the negative value of a numeric expression. Or find the difference between two numbers.
VBScript Not Operator	Perform logical negation of an expression.
VBScript Or Operator	Perform logical disjunction on two expressions.
VBScript Xor Operator	Perform a logical exclusion on two expressions.

VBScript Operator Precedence

When several operations occur in an expression, each part is evaluated and resolved in a predetermined order called operator precedence. Parentheses can be used to override the order of precedence and force some parts of an expression to be evaluated before other parts. Operations within parentheses are always performed before those outside. Within parentheses, however, normal operator precedence is maintained.

When expressions contain operators from more than one category, arithmetic operators are evaluated first, comparison operators are evaluated next, and logical operators are evaluated last. Comparison operators all have equal precedence; that is, they are evaluated in the left-to-right order in which they appear. Arithmetic and logical operators are evaluated in the following order of precedence:

Arithmetic	**Comparison**	**Logical**
Exponentiation (^)	Equality (=)	Not
Negation (-)	Inequality (<>)	And
Multiplication and division (*, /)	Less than (<)	Or
Integer division (\)	Greater than (>)	Xor
Modulus arithmetic (**Mod**)	Less than or equal to (<=)	Eqv
Addition and subtraction (+, -)	Greater than or equal to (>=)	Imp
String concatenation (**&**)	Is	&

When multiplication and division occur together in an expression, each operation is evaluated as it occurs from left to right. Likewise, when addition and subtraction occur together in an expression, each operation is evaluated in order of appearance from left to right.

The string concatenation operator (**&**) is not an arithmetic operator, but in precedence it does fall after all arithmetic operators and before all comparison operators. The **Is** operator is an object reference comparison operator. It does not compare objects or their values; it checks only to determine if two object references refer to the same object.

VBScript Addition Operator (+)

Used to sum two numbers.

Syntax: VBScript Addition Operator (+)

```
result = expression1 + expression2
```

Arguments: VBScript Addition Operator (+)

result

Any numeric variable.

expression1

Any expression.

expression2

Any expression.

Remarks: VBScript Addition Operator (+)

Although you can also use the + operator to concatenate two character strings, you should use the **&** operator for concatenation to eliminate ambiguity and provide self-documenting code.

When you use the + operator, you may not be able to determine whether addition or string concatenation will occur.

The underlying subtype of the expressions determines the behavior of the + operator in the following way:

If	Then
Both expressions are numeric	Add.
Both expressions are strings	Concatenate.
One expression is numeric and the other is a string	Add.

If one or both expressions are **Null** expressions, *result* is **Null**. If both expressions are **Empty**, *result* is an **Integer** subtype. However, if only one expression is **Empty**, the other expression is returned unchanged as *result.*

VBScript And Operator

Used to perform a logical conjunction on two expressions.

Syntax: VBScript And Operator

```
result = expression1 And expression2
```

Arguments: VBScript And Operator

result

Any numeric variable.

expression1

Any expression.

expression2

Any expression.

Remarks: VBScript And Operator

If, and only if, both expressions evaluate to **True**, *result* is **True**. If either expression evaluates to **False**, *result* is **False**. The following table illustrates how *result* is determined:

If *expression1* is	And *expression2* is	The *result* is
True	True	True
True	False	False
True	Null	Null
False	True	False
False	False	False
False	Null	False
Null	True	Null
Null	False	False
Null	Null	Null

The **And** operator also performs a bitwise comparison of identically positioned bits in two numeric expressions and sets the corresponding bit in *result* according to the following table:

If bit in *expression1* is	And bit in *expression2* is	The *result* is
0	0	0
0	1	0
1	0	0
1	1	1

VBScript Assignment Operator (=)

Used to assign a value to a variable or property.

Syntax: VBScript Assignment Operator (=)

```
variable = value
```

Arguments: VBScript Assignment Operator (=)

variable

Any variable or any writable property.

value

Any numeric or string literal, constant, or expression.

Remarks: VBScript Assignment Operator (=) [VBScript Assignment Operator ('equals')]

The name on the left side of the equal sign can be a simple scalar variable or an element of an array. Properties on the left side of the equal sign can only be those properties that are writable at run time.

VBScript Concatenation Operator (&)

Used to force string concatenation of two expressions.

Syntax: VBScript Concatenation Operator (&)

```
result = expression1 & expression2
```

Arguments: VBScript Concatenation Operator (&)

result

Any variable.

expression1

Any expression.

expression2

Any expression.

Remarks: VBScript Concatenation Operator (&)

Whenever an expression is not a string, it is converted to a **String** subtype. If both expressions are **Null**, *result* is also **Null**. However, if only one expression is **Null**, that expression is treated as a zero-length string ("") when concatenated with the other expression. Any expression that is **Empty** is also treated as a zero-length string.

VBScript Division Operator (/)

Used to divide two numbers and return a floating-point result.

Syntax: VBScript Division Operator (/)

```
result = number1/number2
```

Arguments: VBScript Division Operator (/)

result

Any numeric variable.

number1

Any numeric expression.

number2

Any numeric expression.

Remarks: VBScript Division Operator (/)

If one or both expressions are **Null** expressions, *result* is **Null**. Any expression that is **Empty** is treated as 0.

VBScript Eqv Operator

Used to perform a logical equivalence on two expressions.

Syntax: VBScript Eqv Operator

```
result = expression1 Eqv expression2
```

Arguments: VBScript Eqv Operator

result

any numeric variable.

expression1

Any expression.

expression2

Any expression.

Remarks: VBScript Eqv Operator

If either expression is **Null**, *result* is also **Null**. When neither expression is **Null**, *result* is determined according to the following table:

If *expression1* is	And *expression2* is	The *result* is
True	True	True
True	False	False
False	True	False
False	False	True

The **Eqv** operator performs a bitwise comparison of identically positioned bits in two numeric expressions and sets the corresponding bit in *result* according to the following table:

If bit in *expression1* is	And bit in *expression2* is	The *result* is
0	0	1
0	1	0
1	0	0
1	1	1

VBScript Exponentiation Operator (^)

Used to raise a number to the power of an exponent.

Syntax: VBScript Exponentiation Operator (^)

```
result = number^exponent
```

Arguments: VBScript Exponentiation Operator (^)

result

Any numeric variable.

number

Any numeric expression.

exponent

Any numeric expression.

Remarks: VBScript Exponentiation Operator (^)

Number can be negative only if *exponent* is an integer value. When more than one exponentiation is performed in a single expression, the ^ operator is evaluated as it is encountered from left to right.

If either *number* or *exponent* is a **Null** expression, result is also **Null**.

VBScript Imp Operator

Used to perform a logical implication on two expressions.

Syntax: VBScript Imp Operator

```
result = expression1 Imp expression2
```

Arguments: VBScript Imp Operator

result

Any numeric variable.

expression1

Any expression.

expression2

Any expression.

Remarks: VBScript Imp Operator

The following table illustrates how *result* is determined:

If *expression1* is	And *expression2* is	Then *result* is
True	True	True
True	False	False
True	Null	Null
False	True	True
False	False	True
False	Null	True
Null	True	True
Null	False	Null
Null	Null	Null

The **Imp** operator performs a bitwise comparison of identically positioned bits in two numeric expressions and sets the corresponding bit in *result* according to the following table:

If bit in *expression1* is	And bit in *expression2* is	Then *result* is
0	0	1
0	1	1
1	0	0
1	1	1

VBScript Integer Division Operator (\)

Used to divide two numbers and returns an integer result.

Syntax: VBScript Integer Division Operator (\)

```
result = number1\number2
```

Arguments: VBScript Integer Division Operator (\)

result

Any numeric variable.

number1

Any numeric expression.

number2

Any numeric expression.

Remarks: VBScript Integer Division Operator (\)

Before division is performed, numeric expressions are rounded to **Byte**, **Integer**, or **Long** subtype expressions.

If any expression is **Null**, *result* is also **Null**. Any expression that is **Empty** is treated as 0.

VBScript Is Operator

Used to compare two object reference variables.

Syntax: VBScript Is Operator

```
result = object1 Is object2
```

Arguments: VBScript Is Operator

result

Any numeric variable.

object1

Any object name.

object2

Any object name.

Remarks: VBScript Is Operator

If *object1* and *object2* both refer to the same object, *result* is **True**; if they do not, *result* is **False**. Two variables can be made to refer to the same object in several ways.

In the following example, A has been set to refer to the same object as B:

```
Set A = B
```

The following example makes A and B refer to the same object as C:

```
Set A = C
Set B = C
```

VBScript Mod Operator

Used to divide two numbers and return only the remainder.

Syntax: VBScript Mod Operator

```
result = number1 Mod number2
```

Arguments: VBScript Mod Operator

result

Any numeric variable.

number1

Any numeric expression.

number2

Any numeric expression.

Remarks: VBScript Mod Operator

The modulus, or remainder, operator divides *number1* by *number2* (rounding floating-point numbers to integers) and returns only the remainder as *result.* For example, in the following expression, A (which is *result*) equals 5.

```
A = 19 Mod 6.7
```

If any expression is **Null**, result is also **Null**. Any expression that is **Empty** is treated as 0.

VBScript Multiplication Operator (*)

Used to multiply two numbers.

Syntax: VBScript Multiplication Operator (*)

```
result = number1*number2
```

Arguments: VBScript Multiplication Operator (*)

result

Any numeric variable.

number1

Any numeric expression.

number2

Any numeric expression.

Remarks: VBScript Multiplication Operator (*)

If one or both expressions are **Null** expressions, result is **Null**. If an expression is **Empty**, it is treated as if it were 0.

VBScript Negation and Subtraction Operator (-)

Used to find the difference between two numbers or to indicate the negative value of a numeric expression.

Syntax: VBScript Negation and Subtraction Operator (-) 1

```
result = number1-number2
```

Syntax: VBScript Negation and Subtraction Operator (-) 2

```
-number
```

Arguments: VBScript Negation and Subtraction Operator (-)

result

Any numeric variable.

number

Any numeric expression.

number1

Any numeric expression.

number2

Any numeric expression.

Remarks: VBScript Negation and Subtraction Operator (-)

In Syntax 1, the **-** operator is the arithmetic subtraction operator used to find the difference between two numbers. In Syntax 2, the **-** operator is used as the unary negation operator to indicate the negative value of an expression.

If one or both expressions are **Null** expressions, *result* is **Null**. If an expression is **Empty**, it is treated as if it were 0.

VBScript Not Operator

Used to perform logical negation on an expression.

Syntax: VBScript Not Operator

```
result = Not expression
```

Arguments: VBScript Not Operator

result

Any numeric variable.

expression

Any expression.

Remarks: VBScript Not Operator

The following table illustrates how *result* is determined:

If *expression* is	Then *result* is
True	False
False	True
Null	Null

In addition, the **Not** operator inverts the bit values of any variable and sets the corresponding bit in *result* according to the following table:

Bit in *expression*	Bit in *result*
0	1
1	0

VBScript Or Operator

Used to perform a logical disjunction on two expressions.

Syntax: VBScript Or Operator

```
result = expression1 Or expression2
```

Arguments: VBScript Or Operator

result

Any numeric variable.

expression1

Any expression.

expression2

Any expression.

Remarks: VBScript Or Operator

If either or both expressions evaluate to **True**, *result* is **True**. The following table illustrates how *result* is determined:

If *expression1* is	And *expression2* is	Then *result* is
True	True	True
True	False	True
True	Null	True
False	True	True
False	False	False
False	Null	Null

Null	True	True
Null	False	Null
Null	Null	Null

The **Or** operator also performs a bitwise comparison of identically positioned bits in two numeric expressions and sets the corresponding bit in *result* according to the following table:

If bit in *expression1* is	And bit in *expression2* is	Then *result* is
0	0	0
0	1	1
1	0	1
1	1	1

VBScript Xor Operator

Used to perform a logical exclusion on two expressions.

Syntax: VBScript Xor Operator

```
result = expression1 Xor expression2
```

Arguments: VBScript Xor Operator

result

Any numeric variable.

expression1

Any expression.

expression2

Any expression.

Remarks: VBScript Xor Operator

If one, and only one, of the expressions evaluates to **True**, *result* is **True**. However, if either expression is **Null**, *result* is also **Null**. When neither expression is **Null**, *result* is determined according to the following table:

If *expression1* is	And *expression2* is	Then *result* is
True	True	False
True	False	True
False	True	True
False	False	False

The **Xor** operator also performs a bitwise comparison of identically positioned bits in two numeric expressions and sets the corresponding bit in *result* according to the following table:

If bit in *expression1* is	And bit in *expression2* is	Then *result* is
0	0	0
0	1	1
1	0	1
1	1	0

VBScript Statements

VBScript Statements [does this belong here?]

Statement	Description
VBScript Call Statement	Transfers control to a **Sub** procedure or **Function** procedure.
VBScript Const Statement	Declares constants for use in place of literal values.
VBScript Dim Statement	Declares variables and allocates storage space.
VBScript Do. . . Loop Statement	Repeats a block of statements while a condition is **True** or until a condition becomes **True**.
VBScript Erase Statement	Reinitializes fixed-size arrays and deallocates dynamic array storage space.
VBScript Exit Statement	Exits a block of **Do…Loop**, **For…Next**, **Function**, or **Sub** code.
VBScript For. . . Next Statement	Repeats a group of statements a specified number of times.
VBScript For Each. . . Next Statement	Repeats a group of statements for each element in an array or collection.
VBScript Function Statement	Declares the name, arguments, and code that form the body of a **Function** procedure.
VBScript If. . . Then. . . Else Statement	Conditionally executes a group of statements.
VBScript On Error Statement	Enables error-handling.
VBScript Option Explicit Statement	Used at script level to force explicit declaration of all variables in that script.
VBScript Private Statement	Used at script level to declare private variables and

	allocate storage space.
VBScript Public Statement	Used at script level to declare public variables and allocate storage space.
VBScript Randomize Statement	Initializes the random-number generator.
VBScript ReDim Statement	Used at procedure level to declare dynamic-array variables and allocate or reallocate storage space.
VBScript Rem Statement	Used to include explanatory remarks and comments in a script.
VBScript Select Case Statement	Executes one of several groups of statements based on the value of an expression.
VBScript Set Statement	Assigns an object reference to a variable or property.
VBScript Sub Statement	Declares the name, arguments, and code that form the body of a **Sub** procedure.
VBScript While. . . Wend Statement	Executes a series of statements as long as a given condition is **True**.

VBScript Call Statement

Transfers control to a **Sub** procedure or **Function** procedure.

Syntax: VBScript Call Statement

```
[Call] name [argumentslist]
```

Arguments: VBScript Call Statement[0]

Call

A keyword. If specified, you must enclose *argumentslist* in parentheses. Optional. For example:

```
Call MyProc (0)
```

name

The name of the procedure to call. Required.

argumentslist

A comma-delimited list of variables, arrays, or expressions to pass to the procedure. Optional.

Remarks: VBScript Call Statement

You are not required to use the **Call** keyword when calling a procedure. However, if you use the **Call** keyword to call a procedure that requires arguments, *argumentslist* must be enclosed in parentheses. If you omit the **Call** keyword, you also must omit the parentheses around *argumentslist*. If you use either **Call** syntax to call any intrinsic or user-defined function, the function's return value is discarded.

VBScript Const Statement

Declares constants for use in place of literal values.

Syntax: VBScript Const Statement

```
[Public | Private] Const constname = expression
```

Arguments: VBScript Const Statement

Public

A keyword used at script level to declare constants that are available to all procedures in all scripts. Not allowed in procedures. Optional.

Private

A keyword used at script level to declare constants that are available only within the script where the declaration is made. Not allowed in procedures. Optional.

constname

The name of the constant; follows standard variable naming conventions. Required.

expression

A literal or other constant, or any combination that includes all arithmetic or logical operators except **Is.** Required.

Remarks: VBScript Const Statement

Constants are public by default. Within procedures, constants are always private; their visibility can't be changed. Within a script, the default visibility of a script-level constant can be changed using the **Private** keyword.

To combine several constant declarations on the same line, separate each constant assignment with a comma. When constant declarations are combined in this way, the **Public** or **Private** keyword, if used, applies to all of them.

You can't use variables, user-defined functions, or intrinsic VBScript functions (such as **Chr**) in constant declarations. By definition, they can't be constants. You also can't create a constant from any expression that involves an operator; that is, only simple constants are allowed. Constants declared in a **Sub** or **Function** procedure are local to that procedure. A constant declared outside a procedure is defined throughout the script in which it is declared. You can use constants anywhere you can use an expression. The following code illustrates the use of the **Const** statement:

```
Const MyVar = 459      ' Constants are Public by default.

Private Const MyString = "HELP"      ' Declare Private constant.

Const MyStr = "Hello", MyNumber = 3.4567      ' Declare multiple
constants on same line.
```

Tip

Constants can make your scripts self-documenting and easy to modify. Unlike variables, constants can't be inadvertently changed while your script is running.

VBScript Dim Statement

Declares variables and allocates storage space.

Syntax: VBScript Dim Statement

```
Dim varname[([subscripts])][, varname[([subscripts])]] . . .
```

Arguments: VBScript Dim Statement

varname

Name of the variable; follows standard variable naming conventions.

subscripts

Dimensions of an array variable; up to 60 multiple dimensions may be declared. The subscripts argument uses the following syntax:

upperbound [,upperbound] . . .

The lower bound of an array is always zero.

Remarks: VBScript Dim Statement

Variables declared with **Dim** at the script level are available to all procedures within the script. At the procedure level, variables are available only within the procedure.

You can also use the **Dim** statement with empty parentheses to declare a dynamic array. After declaring a dynamic array, use the **ReDim** statement within a procedure to define the number of dimensions and elements in the array. If you try to redeclare a dimension for an array variable whose size was explicitly specified in a **Dim** statement, an error occurs.

When variables are initialized, a numeric variable is initialized to 0 and a string is initialized to a zero-length string (""). The following examples illustrate the use of the **Dim** statement:

```
Dim Names(9)              ' Declare an array with 10 elements.

Dim Names()               ' Declare a dynamic array.

Dim MyVar, MyNum      ' Declare two variables.
```

Tip

When you use the **Dim** statement in a procedure, you generally put the **Dim** statement at the beginning of the procedure.

VBScript Do. . . Loop Statement

Repeats a block of statements while a condition is **True** or until a condition becomes **True**.

Syntax: VBScript Do. . . Loop Statement

```
Do [{While | Until} condition]
[statements]
[Exit Do]
[statements]
Loop
```

Or, you can use this syntax:

```
Do
[statements]
[Exit Do]
[statements]
Loop [{While | Until} condition]
```

Arguments: VBScript Do. . . Loop Statement

condition

A numeric or string expression that is **True** or **False**. If *condition* is **Null**, *condition* is treated as False.

statements

One or more statements that are repeated while or until condition is **True**.

Remarks: VBScript Do. . . Loop Statement

The **Exit** statement can only be used within a **Do...Loop** control structure to provide an alternate way to exit a **Do...Loop**. Any number of **Exit Do** statements may be placed anywhere in the **Do...Loop**. Often used with the evaluation of some condition (for example, **If…Then…Else**), **Exit Do** transfers control to the statement immediately following the Loop.

When used within nested **Do...Loop** statements, **Exit Do** transfers control to the loop that is one nested level above the loop where it occurs.

The following examples illustrate use of the Do...Loop statement:

```
Do Until DefResp = vbNo
      MyNum = Int (6 * Rnd + 1)      ' Generate a random integer
between 1 and 6.
      DefResp = MsgBox (MyNum & " Do you want another number?",
vbYesNo)
Loop
```

```
Dim Check, Counter

Check = True: Counter = 0      ' Initialize variables.

Do                                                    ' Outer
loop.

      Do While Counter < 20            ' Inner loop.

            Counter = Counter + 1      ' Increment Counter.

            If Counter = 10 Then        ' If condition is True...

                  Check = False                ' set value of flag
to False.

                  Exit Do                            ' Exit inner
loop.

            End If

      Loop

Loop Until Check = False            ' Exit outer loop immediately.
```

VBScript Erase Statement

Reinitializes the elements of fixed-size arrays and deallocates dynamic-array storage space.

Syntax: VBScript Erase Statement

```
Erase array
```

Arguments: VBScript Erase Statement

array

The name of the array variable to be erased.

Remarks: VBScript Erase Statement

It is important to know whether an array is fixed-size (ordinary) or dynamic because **Erase** behaves differently depending on the type of array. **Erase** recovers no memory for fixed-size arrays. **Erase** sets the elements of a fixed array as follows:

Type of array	Effect of Erase on fixed-array elements
Fixed numeric array	Sets each element to zero.
Fixed string array	Sets each element to zero-length ("").
Array of objects	Sets each element to the special value Nothing.

Erase frees the memory used by dynamic arrays. Before your program can refer to the dynamic array again, it must redeclare the array variable's dimensions using a **ReDim** statement.

The following example illustrates the use of the **Erase** statement.

```
Dim NumArray(9)
Dim DynamicArray()
ReDim DynamicArray(9)      ' Allocate storage space.
Erase NumArray      ' Each element is reinitialized.
Erase DynamicArray      ' Free memory used by array.
```

VBScript Exit Statement

Exits a block of **Do...Loop**, **For...Next**, **Function**, or **Sub** code.

Syntax: VBScript Exit Statement

```
Exit Do
Exit For
Exit Function
Exit Sub
```

Exit Do

Provides a way to exit a **Do…Loop** statement. It can be used only inside a **Do...Loop** statement. **Exit Do** transfers control to the statement following the Loop statement. When used within nested **Do...Loop** statements, **Exit Do** transfers control to the loop that is one nested level above the loop where it occurs.

Exit For

Provides a way to exit a **For** loop. It can be used only in a **For…Next** or **For Each…Next** loop. Exit For transfers control to the statement following the **Next** statement. When used within nested **For** loops, **Exit For** transfers control to the loop that is one nested level above the loop where it occurs.

Exit Function

Immediately exits the **Function** procedure in which it appears. Execution continues with the statement following the statement that called the **Function**.

Exit Sub

Immediately exits the **Sub** procedure in which it appears. Execution continues with the statement following the statement that called the **Sub**.

The following example illustrates the use of the **Exit** statement:

```
Sub RandomLoop
      Dim I, MyNum
      Do      ' Set up infinite loop.
            For I = 1 To 1000      ' Loop 1000 times.
```

```
                    MyNum = Int(Rnd * 100)      ' Generate random
numbers.
                    Select Case MyNum      ' Evaluate random number.
                          Case 17: MsgBox "Case 17"
                                Exit For      ' If 17, exit
For...Next.
                          Case 29: MsgBox "Case 29"
                                Exit Do      ' If 29, exit Do...Loop.
                          Case 54: MsgBox "Case 54"
                                Exit Sub      ' If 54, exit Sub
procedure.
                          End Select
             Next
      Loop
End Sub
```

VBScript For. . . Next Statement

Repeats a group of statements a specified number of times.

Syntax: VBScript For. . . Next Statement

For *counter* **=** *start* **To** *end* [**Step** *step*]

[*statements*]

[**Exit For**]

[*statements*]

Next

Arguments: VBScript For. . . Next Statement

counter

A numeric variable used as a loop counter. The variable can't be an array element or an element of a user-defined type.

start

The initial value of *counter*.

end

The final value of *counter*.

step

The amount *counter* is changed each time through the loop. If not specified, *step* defaults to one.

statements

One or more statements between **For** and **Next** that are executed the specified number of times.

Remarks: VBScript For. . . Next Statement

The *step* argument can be either positive or negative. The value of the *step* argument determines loop processing as follows:

Value	Loop executes if
Positive or 0	counter <= end
Negative	counter >= end

Once the loop starts and all statements in the loop have executed, *step* is added to *counter*. At this point, either the statements in the loop execute again (based on the same test that caused the loop to execute initially), or the loop is exited and execution continues with the statement following the **Next** statement.

Tip

Changing the value of *counter* while inside a loop can make it more difficult to read and debug your code.

Exit can only be used within a **For Each…Next** or **For...Next** control structure to provide an alternate way to exit. Any number of **Exit For** statements may be placed anywhere in the loop. **Exit For** is often used with the evaluation of some condition (for example, **If…Then…Else**), and transfers control to the statement immediately following **Next**.

You can nest **For...Next** loops by placing one **For...Next** loop within another. Give each loop a unique variable name as its counter. The following construction is correct:

```
For I = 1 To 10

For J = 1 To 10

For K = 1 To 10

. . .

Next

Next

Next
```

VBScript For Each. . . Next Statement

Repeats a group of statements for each element in an array or collection.

Syntax: VBScript For Each. . . Next Statement

```
For Each element In group
[statements]
[Exit For]
[statements]
Next [element]
```

Arguments: VBScript For Each. . . Next Statement

element

A variable used to iterate through the elements of the collection or array. For collections, *element* can only be a **Variant** variable, a generic **Object** variable, or any specific Automation object variable. For arrays, *element* can only be a **Variant** variable.

group

The name of an object collection or array.

statements

One or more statements that are executed on each item in *group*.

Remarks: VBScript For Each. . . Next Statement

The **For Each** block is entered if there is at least one element in *group*. Once the loop has been entered, all the statements in the loop are executed for the first element in *group.* As long as there are more elements in *group,* the statements in the loop continue to execute for each element. When there are no more elements in *group,* the loop is exited and execution continues with the statement following the **Next** statement.

Exit FOR can only be used within a **For Each…Next** or **For...Next** control structure to provide an alternate way to exit. Any number of **Exit For** statements may be placed anywhere in the loop. **Exit For** is often used with the evaluation of some condition (for example, **If…Then…Else**), and transfers control to the statement immediately following **Next**.

You can nest **For Each...Next** loops by placing one **For Each...Next** loop within another. However, each loop element must be unique.

The following example illustrates use of the **For Each...Next** statement:

```
Function ShowFolderList(folderspec)
      Dim fso, f, f1, fc, s
      Set fso = CreateObject("Scripting.FileSystemObject")
      Set f = fso.GetFolder(folderspec)
      Set fc = f.Files
      For Each f1 in fc
            s = s & f1.name
```

```
            s = s & "<BR>"
        Next
        ShowFolderList = s
End Function
```

Note

If you omit *element* in a **Next** statement, execution continues as if you had included it. If a **Next** statement is encountered before its corresponding **For** statement, an error occurs.

VBScript Function Statement

Declares the name, arguments, and code that form the body of a **Function** procedure.

Syntax: VBScript Function Statement

```
[Public | Private] Function name [(arglist)]
[statements]
[name = expression]
[Exit Function]
[statements]
[name = expression]
End Function
```

Arguments: VBScript Function Statement

Public

The **Function** procedure is accessible to all other procedures in all scripts.

Private

The **Function** procedure is accessible only to other procedures in the script where it is declared **Static**. Indicates that the **Function** procedure's local variables are preserved between calls. The **Static** attribute doesn't affect variables that are declared outside the **Function**, even if they are used in the procedure.

name

The name of the **Function**; follows standard variable naming conventions.

arglist

A list of variables representing arguments that are passed to the **Function** procedure when it is called. Multiple variables are separated by commas.

The *arglist* argument has the following syntax and parts:

```
[ByVal | ByRef] varname[( )]
```

ByVal

The argument is passed by value.

ByRef

The argument is passed by reference.

varname

The name of the variable representing the arguments; follows standard variable naming conventions.

statements

Any group of statements to be executed within the body of the Function procedure.

expression

Returns the value of the **Function.**

Remarks: VBScript Function Statement

If not explicitly specified using either **Public** or **Private**, **Function** procedures are public by default, that is, they are visible to all other procedures in your script. The value of local variables in a **Function** is not preserved between calls to the procedure.

All executable code must be contained in procedures. You can't define a **Function** procedure inside another **Function** or **Sub** procedure.

The **Exit** statement causes an immediate exit from a **Function** procedure. Program execution continues with the statement following the statement that called the **Function** procedure. Any number of **Exit Function** statements can appear anywhere in a **Function** procedure.

Like a **Sub** procedure, a **Function** procedure is a separate procedure that can take arguments, perform a series of statements, and change the values of its arguments. However, unlike a **Sub** procedure, you can use a **Function** procedure on the right side of an expression in the same way you use any intrinsic function, such as **Sqr**, **Cos**, or **Chr**, when you want to use the value returned by the function.

Call a **Function** procedure in an expression using the function name followed by the argument list in parentheses. See the **Call** statement for specific information on how to call **Function** procedures.

Caution

> **Function** procedures can be recursive; that is, they can call themselves to perform a given task. However, recursion can lead to stack overflow.
>
> To return a value from a function, assign the value to the function name. Any number of such assignments can appear anywhere within the procedure. If no value is assigned to *name,* the procedure returns a default value: a numeric function returns 0 and a string function returns a zero-length string (""). A function that returns an object reference returns Nothing if no object reference is assigned to *name* (using **Set**) within the **Function**.

The following example shows how to assign a return value to a function named BinarySearch. In this case, **False** is assigned to the name to indicate that some value was not found.

```
Function BinarySearch(. . .)

. . .

' Value not found. Return a value of False.

If lower > upper Then

BinarySearch = False

Exit Function

End If

. . .

End Function
```

Variables used in **Function** procedures fall into two categories: those that are explicitly declared within the procedure and those that are not. Variables that are explicitly declared in a procedure (using **Dim** or the equivalent) are always local to the procedure. Variables that are used but not explicitly declared in a procedure are also local unless they are explicitly declared at some higher level outside the procedure.

Caution

A procedure can use a variable that is not explicitly declared in the procedure, but a naming conflict can occur if anything you have defined at the script level has the same name. If your procedure refers to an undeclared variable that has the same name as another procedure, constant, or variable, it is assumed that your procedure is referring to that script-level name. Explicitly declare variables to avoid this kind of conflict. You can use an **Option Explicit** statement to force explicit declaration of variables.

Caution

VBScript may rearrange arithmetic expressions to increase internal efficiency. Avoid using a **Function** procedure in an arithmetic expression when the function changes the value of variables in the same expression.

VBScript If. . . Then. . . Else Statement

Conditionally executes a group of statements, depending on the value of an expression.

Syntax: VBScript If. . . Then. . . Else Statement

If *condition* **Then** *statements* [**Else** *elsestatements*]

Or, you can use the block form syntax:

```
If condition Then
[statements]
[ElseIf condition-n Then
[elseifstatements]] . . .
[Else
[elsestatements]]
End If
```

Arguments: VBScript If. . . Then. . . Else Statement

condition

One or more of the following two types of expressions:

A numeric or string expression that evaluates to **True** or **False**. If *condition* is **Null**, *condition* is treated as **False**.

An expression of the form **TypeOf** *objectname* Is *objecttype*. The *objectname* is any object reference and *objecttype* is any valid object type. The expression is **True** if *objectname* is of the object type specified by *objecttype*; otherwise it is **False**.

statements

One or more statements separated by colons; executed if *condition* is **True**.

condition-n

Same as *condition*.

elseifstatements

One or more statements executed if the associated *condition-n* is **True**.

elsestatements

One or more statements executed if no previous *condition* or *condition-n* expression is **True**.

Remarks: VBScript If. . . Then. . . Else Statement

You can use the single-line form (first syntax) for short, simple tests. However, the block form (second syntax) provides more structure and flexibility than the single-line form and is usually easier to read, maintain, and debug.

With the single-line syntax, it is possible to have multiple statements executed as the result of an **If...Then** decision, but they must all be on the same line and separated by colons, as in the following statement:

```
If A > 10 Then A = A + 1 : B = B + A : C = C + B
```

When executing a block **If** (second syntax), *condition* is tested. If *condition* is **True**, the statements following **Then** are executed. If *condition* is **False**, each **ElseIf** (if any) is evaluated in turn. When a **True** condition is found, the statements following the associated **Then** are executed.

If none of the **ElseIf** statements are **True** (or there are no **ElseIf** clauses), the statements following **Else** are executed. After executing the statements following **Then** or **Else**, execution continues with the statement following **End If**.

The **Else** and **ElseIf** clauses are both optional. You can have as many **ElseIf** statements as you want in a block **If**, but none can appear after the **Else** clause. Block **If** statements can be nested; that is, contained within one another.

What follows the **Then** keyword is examined to determine whether or not a statement is a block **If**. If anything other than a comment appears after **Then** on the same line, the statement is treated as a single-line **If** statement.

A block **If** statement must be the first statement on a line. The block **If** must end with an **End If** statement.

VBScript On Error Statement

Enables error-handling.

Syntax: VBScript On Error Statement

```
On Error Resume Next
```

Remarks: VBScript On Error Statement

If you don't use an **On Error Resume Next** statement, any run-time error that occurs is fatal; that is, an error message is displayed and execution stops.

On Error Resume Next causes execution to continue with the statement immediately following the statement that caused the run-time error, or with the statement immediately following the most recent call out of the procedure containing the **On Error Resume Next** statement. This allows execution to continue despite a run-time error. You can then build the error-handling routine inline within the procedure. An **On Error Resume Next** statement becomes inactive when another procedure is called, so you should execute an **On Error Resume Next** statement in each called routine if you want inline error-handling within that routine.

The following example illustrates use of the **On Error Resume Next** statement:

```
On Error Resume Next

Err.Raise 6      ' Raise an overflow error.

MsgBox "Error # " & CStr(Err.Number) & " " & Err.Description

Err.Clear      ' Clear the error.
```

VBScript Option Explicit Statement

Used at script level to force explicit declaration of all variables in that script.

Syntax: VBScript Option Explicit Statement

```
Option Explicit
```

Remarks: VBScript Option Explicit Statement

If used, the **Option Explicit** statement must appear in a script before any procedures.

When you use the **Option Explicit** statement, you must explicitly declare all variables using the **Dim**, **Private**, **Public**, or **ReDim** statements. If you attempt to use an undeclared variable name, an error occurs.

The following example illustrates use of the **Option Explicit** statement:

```
Option Explicit      ' Force explicit variable declaration.

Dim MyVar      ' Declare variable.

MyInt = 10      ' Undeclared variable generates error.

MyVar = 10      ' Declared variable does not generate error.
```

Tip

> Use **Option Explicit** to avoid incorrectly typing the name of an existing variable or to avoid confusion in code where the scope of the variable is not clear.

VBScript Private Statement

Used at script level to declare private variables and allocate storage space.

Syntax: VBScript Private Statement

```
Private varname[([subscripts])][, varname[([subscripts])]] . . .
```

Arguments: VBScript Private Statement

varname

The name of the variable; follows standard variable naming conventions.

subscripts

The dimensions of an array variable; up to 60 multiple dimensions may be declared. The *subscripts* argument uses the following syntax:

upper [**,** upper] . . .

The lower bound of an array is always zero.

Remarks: VBScript Private Statement

Private variables are available only to the script in which they are declared.

A variable that refers to an object must be assigned an existing object using the **Set** statement before it can be used. Until it is assigned an object, the declared object variable has the special value Nothing.

You can also use the **Private** statement with empty parentheses to declare a dynamic array. After declaring a dynamic array, use the **ReDim** statement within a procedure to define the number of

dimensions and elements in the array. If you try to redeclare a dimension for an array variable whose size was explicitly specified in a **Private**, **Public**, or **Dim** statement, an error occurs.

When variables are initialized, a numeric variable is initialized to 0 and a string is initialized to a zero-length string ("").

The following example illustrates use of the **Option Explicit** statement:

```
Option Explicit      ' Force explicit variable declaration.
Dim MyVar      ' Declare variable.
MyInt = 10      ' Undeclared variable generates error.
MyVar = 10      ' Declared variable does not generate error.
```

Tip

When you use the **Private** statement in a procedure, you generally put the **Private** statement at the beginning of the procedure.

VBScript Public Statement

Used at script level to declare public variables and allocate storage space.

Syntax: VBScript Public Statement

```
Public varname[([subscripts])][, varname[([subscripts])]] . . .
```

Arguments: VBScript Public Statement

varname

The name of the variable; follows standard variable naming conventions.

subscripts

The dimensions of an array variable; up to 60 multiple dimensions may be declared. The *subscripts* argument uses the following syntax:

upper [**,**upper] . . .

The lower bound of an array is always zero.

Remarks: VBScript Public Statement

Variables declared using the **Public** statement are available to all procedures in all scripts in all projects.

A variable that refers to an object must be assigned an existing object using the **Set** statement before it can be used. Until it is assigned an object, the declared object variable has the special value **Nothing**.

You can also use the **Public** statement with empty parentheses to declare a dynamic array. After declaring a dynamic array, use the **ReDim** statement within a procedure to define the number of

dimensions and elements in the array. If you try to redeclare a dimension for an array variable whose size was explicitly specified in a **Private**, **Public**, or **Dim** statement, an error occurs.

When variables are initialized, a numeric variable is initialized to 0 and a string is initialized to a zero-length string ("").

The following example illustrates the use of the **Public** statement:

```
Public MyNumber      ' Public Variant variable.
Public MyArray(9)      ' Public array variable.
      ' Multiple Public declarations of Variant variables.
Public MyNumber, MyVar, YourNumber
```

VBScript Randomize Statement

Initializes the random-number generator.

Syntax: VBScript Randomize Statement

```
Randomize [number]
```

Arguments: VBScript Randomize Statement

number

Any valid numeric expression.

Remarks: VBScript Randomize Statement

Randomize uses *number* to initialize the **Rnd** function random-number generator, giving it a new seed value. If you omit *number,* the value returned by the system timer is used as the new seed value.

If **Randomize** is not used, the **Rnd** function (with no arguments) uses the same number as a seed the first time it is called, and thereafter uses the last generated number as a seed value.

The following example illustrates use of the **Randomize** statement.

```
Dim MyValue, Response
Randomize      ' Initialize random-number generator.
Do Until Response = vbNo
      MyValue = Int((6 * Rnd) + 1)      ' Generate random value
between 1 and 6.
      MsgBox MyValue
      Response = MsgBox ("Roll again? ", vbYesNo)
Loop
```

Note

To repeat sequences of random numbers, call **Rnd** with a negative argument immediately before using **Randomize** with a numeric argument. Using **Randomize** with the same value for *number* does not repeat the previous sequence.

VBScript ReDim Statement

Used at procedure level to declare dynamic-array variables and allocate or reallocate storage space.

Syntax: VBScript ReDim Statement

```
ReDim [Preserve] varname(subscripts) [, varname(subscripts) . . .
```

Arguments: VBScript ReDim Statement

Preserve

Preserves the data in an existing array when you change the size of the last dimension.

varname

The name of the variable; follows standard variable naming conventions.

subscripts

The dimensions of an array variable; up to 60 multiple dimensions may be declared. The *subscripts* argument uses the following syntax:

upper [**,**upper] . . .

The lower bound of an array is always zero.

Remarks: VBScript ReDim Statement

The **ReDim** statement is used to size or resize a dynamic array that has already been formally declared using a **Private**, **Public**, or **Dim** statement with empty parentheses (without dimension subscripts). You can use the **ReDim** statement repeatedly to change the number of elements and dimensions in an array.

If you use the **Preserve** keyword, you can resize only the last array dimension, and you can't change the number of dimensions at all. For example, if your array has only one dimension, you can resize that dimension because it is the last and only dimension. However, if your array has two or more dimensions, you can change the size of only the last dimension and still preserve the contents of the array.

The following example shows how you can increase the size of the last dimension of a dynamic array without erasing any existing data contained in the array.

```
ReDim X(10, 10, 10)

. . .

ReDim Preserve X(10, 10, 15)
```

Caution

If you make an array smaller than it was originally, data in the eliminated elements is lost.

When variables are initialized, a numeric variable is initialized to 0 and a string variable is initialized to a zero-length string (""). A variable that refers to an object must be assigned an existing object using the **Set** statement before it can be used. Until it is assigned an object, the declared object variable has the special value **Nothing**.

VBScript Rem Statement

Used to include explanatory remarks in a program.

Syntax: VBScript Rem Statement

```
Rem comment
```

or

```
' comment
```

Arguments: VBScript Rem Statement

comment

The text of any comment you want to include. After the **Rem** keyword, a space is required before *comment.*

Remarks: VBScript Rem Statement

As shown in the syntax section, you can use an apostrophe (**'**) instead of the **Rem** keyword. If the **Rem** keyword follows other statements on a line, it must be separated from the statements by a colon. However, when you use an apostrophe, the colon is not required after other statements.

The following example illustrates the use of the **Rem** statement.

```
Dim MyStr1, MyStr2

MyStr1 = "Hello" : Rem Comment after a statement separated by a
colon.

MyStr2 = "Goodbye"  ' This is also a comment; no colon is needed.

Rem Comment on a line with no code; no colon is needed.
```

VBScript Select Case Statement

Executes one of several groups of statements, depending on the value of an expression.

Syntax: VBScript Select Case Statement

```
Select Case testexpression

[Case expressionlist-n
```

```
[statements-n]] . . .
[Case Else expressionlist-n
[elsestatements-n]]
End Select
```

Arguments: VBScript Select Case Statement

testexpression

Any numeric or string expression.

expressionlist-n

Required if **Case** appears. Delimited list of one or more expressions.

statements-n

One or more statements executed if *testexpression* matches any part of *expressionlist-n.*

elsestatements

One or more statements executed if *testexpression* doesn't match any of the **Case** clauses.

Remarks: VBScript Select Case Statement

If *testexpression* matches any **Case** *expressionlist* expression, the statements following that **Case** clause are executed up to the next **Case** clause, or for the last clause, up to **End Select**. Control then passes to the statement following **End Select**. If *testexpression* matches an *expressionlist* expression in more than one **Case** clause, only the statements following the first match are executed.

The **Case Else** clause is used to indicate the *elsestatements* to be executed if no match is found between the *testexpression* and an *expressionlist* in any of the other **Case** selections. Although not required, it is a good idea to have a **Case Else** statement in your **Select Case** block to handle unforeseen *testexpression* values. If no **Case** *expressionlist* matches *testexpression* and there is no **Case Else** statement, execution continues at the statement following **End Select**.

Select Case statements can be nested. Each nested **Select Case** statement must have a matching **End Select** statement.

The following example illustrates the use of the **Select Case** statement:

```
Dim Color, MyVar
Sub ChangeBackground (Color)
      MyVar = lcase (Color)
      Select Case MyVar
            Case "red"          document.bgColor = "red"
            Case "green"      document.bgColor = "green"
            Case "blue"        document.bgColor = "blue"
            Case Else            MsgBox "pick another color"
```

```
        End Select
End Sub
```

VBScript Set Statement

Assigns an object reference to a variable or property.

Syntax: VBScript Set Statement

```
Set objectvar = {objectexpression | Nothing}
```

Arguments: VBScript Set Statement

objectvar

The name of the variable or property; follows standard variable naming conventions.

objectexpression

An expression consisting of the name of an object, another declared variable of the same object type, or a function or method that returns an object of the same object type.

Nothing

Discontinues association of *objectvar* with any specific object. Assigning *objectvar* to **Nothing** releases all the system and memory resources associated with the previously referenced object when no other variable refers to it.

Remarks: VBScript Set Statement

To be valid, *objectvar* must be an object type consistent with the object being assigned to it.

The **Dim**, **Private**, **Public**, or **ReDim** statements only declare a variable that refers to an object. No actual object is referred to until you use the **Set** statement to assign a specific object.

Generally, when you use **Set** to assign an object reference to a variable, no copy of the object is created for that variable. Instead, a reference to the object is created. More than one object variable can refer to the same object. Because these variables are references to (rather than copies of) the object, any change in the object is reflected in all variables that refer to it.

```
Function ShowFreeSpace(drvPath)
      Dim fso, d, s
      Set fso = CreateObject("Scripting.FileSystemObject")
      Set d = fso.GetDrive(fso.GetDriveName(drvPath))
      s = "Drive " & UCase(drvPath) & " - "
      s = s & d.VolumeName  & "<BR>"
      s = s & "Free Space: " & FormatNumber(d.FreeSpace/1024, 0)
      s = s & " Kbytes"
      ShowFreeSpace = s
```

```
End Function
```

Using the **New** keyword allows you to concurrently create an instance of a class and assign it to an object reference variable. The variable to which the instance of the class is being assigned must already have been declared with the **Dim** (or equivalent) statement.

Refer to the documentation for the **GetRef** function for information on using **Set** to associate a procedure with an event.

VBScript Sub Statement

Declares the name, arguments, and code that form the body of a **Sub** procedure.

Syntax: VBScript Sub Statement

```
[Public | Private] Sub name [(arglist)]
[statements]
[Exit Sub]
[statements]
End Sub
```

Arguments: VBScript Sub Statement

Public

The **Sub** procedure is accessible to all other procedures in all scripts.

Private

The **Sub** procedure is accessible only to other procedures in the script where it is declared

name

The name of the **Sub** procedure; follows standard variable naming conventions.

arglist

A list of variables representing arguments that are passed to the **Sub** procedure when it is called. Multiple variables are separated by commas.

```
[ByVal | ByRef] varname[( )]
```

ByVal

The argument is passed by value.

ByRef

The argument is passed by reference.

varname

The name of the variable representing the argument; follows standard variable naming conventions.

statements

Any group of statements to be executed within the body of the **Sub** procedure.

Remarks: VBScript Sub Statement

If not explicitly specified using either **Public** or **Private**, **Sub** procedures are public by default, that is, they are visible to all other procedures in your script. The value of local variables in a **Sub** procedure is not preserved between calls to the procedure.

All executable code must be contained in procedures. You can't define a **Sub** procedure inside another **Sub** or **Function** procedure.

The **Exit** statement causes an immediate exit from a **Sub** procedure. Program execution continues with the statement following the statement that called the **Sub** procedure. Any number of **Exit Sub** statements can appear anywhere in a **Sub** procedure.

Like a **Function** procedure, a **Sub** procedure is a separate procedure that can take arguments, perform a series of statements, and change the value of its arguments. However, unlike a **Function** procedure, which returns a value, a **Sub** procedure can't be used in an expression.

You call a **Sub** procedure using the procedure name followed by the argument list. See the **Call** statement for specific information on how to call **Sub** procedures.

Caution

Sub procedures can be recursive; that is, they can call themselves to perform a given task. However, recursion can lead to stack overflow. The **Static** keyword usually is not used with recursive **Sub** procedures.

Variables used in **Sub** procedures fall into two categories: those that are explicitly declared within the procedure and those that are not. Variables that are explicitly declared in a procedure (using **Dim** or the equivalent) are always local to the procedure. Variables that are used but not explicitly declared in a procedure are also local unless they are explicitly declared at some higher level outside the procedure.

Caution

A procedure can use a variable that is not explicitly declared in the procedure, but a naming conflict can occur if anything you have defined at the script level has the same name. If your procedure refers to an undeclared variable that has the same name as another procedure, constant or variable, it is assumed that your procedure is referring to that script-level name. Explicitly declare variables to avoid this kind of conflict. You can use an **Option Explicit** statement to force explicit declaration of variables.

VBScript While. . . Wend Statement

Executes a series of statements as long as a given condition is **True**.

Syntax: VBScript While. . . Wend Statement

```
While condition
```

```
[statements]
Wend
```

Arguments: VBScript While. . . Wend Statement

condition

A numeric or string expression that evaluates to **True** or **False**. If *condition* is **Null**, *condition* is treated as **False**.

statements

One or more statements executed while *condition* is **True**.

Remarks: VBScript While. . . Wend Statement

If *condition* is **True**, all statements in statements are executed until the **Wend** statement is encountered. Control then returns to the **While** statement and *condition* is again checked. If *condition* is still **True**, the process is repeated. If it is not **True**, execution resumes with the statement following the **Wend** statement.

While...Wend loops can be nested to any level. Each **Wend** matches the most recent **While**.

The following example illustrates use of the **While...Wend** statement:

```
Dim Counter
Counter = 0      ' Initialize variable.
While Counter < 20      ' Test value of Counter.
      Counter = Counter + 1      ' Increment Counter.
      Alert Counter
Wend      ' End While loop when Counter > 19.
```

Tip

The **Do…Loop** statement provides a more structured and flexible way to perform looping.

VBScript Functions

Note

The behavior of VBScript functions such as **MidB**, **ChrB**, **LeftB**, and **AscB** depends on the byte ordering of the hardware platform, and the number of bytes used to represent Unicode characters in the system software. The functions will produce different results on different operating systems. The behavior described in this section pertains to the Win32 version.

VBScript Functions

Function	Description
VBScript Abs Function	Returns the absolute value of a number.
VBScript Array Function	Returns an array.
VBScript Asc Function	Returns the ASNI character code corresponding to the first letter in a string.
VBScript Atn Function	Returns the arctangent of a number.
VBScript CBool Function	Returns an expression that has been converted to a **Boolean** variant.
VBScript CByte Function	Returns an expression that has been converted to a **Byte** variant.
VBScript CCur Function	Returns an expression that has been converted to **Currency** variant.
VBScript CDate Function	Returns an expression that has been converted to a **Date** variant.
VBScript CDbl Function	Returns an expression that has been converted to a **Double** variant.
VBScript Chr Function	Returns the character associated with the specified ANSI character code.
VBScript CInt Function	Returns an expression that has been converted to an **Integer** variant.
VBScript CLng Function	Returns an expression that has been converted to a **Long** variant.
VBScript Cos Function	Returns the cosine of an angle.
VBScript CreateObject Function	Creates and returns a reference to an Automation object.
VBScript CSng Function	Returns an object that has been converted to a **Single** variant.
VBScript CStr Function	Returns an expression that has been converted to a **String** variant.
VBScript Date Function	Returns the current system date.
VBScript DateAdd Function	Returns a date to which a specified time interval has been added.
VBScript DateDiff Function	Returns the number of intervals between two dates.
VBScript DatePart Function	Returns the specified part of a given date.
VBScript DateSerial Function	Returns a **Date** variant for a specified year, month and day.
VBScript DateValue Function	Returns a **Date** variant.

VBScript Day Function	Returns the day of the month (1-31).
VBScript Exp Function	Returns *e* raised to a power.
VBScript Filter Function	Returns a subset of a string array based on a specified filter criteria.
VBScript Fix, Int Function	Returns the integer portion of a number.
VBScript FormatCurrency Function	Returns an expression formatted as a currency value.
VBScript FormatDateTime Function	Returns an expression formatted as a date or time.
VBScript FormatNumber Function	Returns an expression formatted as a number.
VBScript FormatPercent Function	Returns an expression formatted as a percentage.
VBScript GetObject Function	Returns a reference to an Automation object from a file. Windows systems only.
VBScript Hex Function	Returns a string representing the hexadecimal value of a number.
VBScript Hour Function	Returns the hour of the day (0-23).
VBScript InputBox Function	Displays a prompt in a dialog box. Client-side only.
VBScript InStr Function	Returns the position of the first occurrence of one string within another.
VBScript InStrRev Function	Returns the position of an occurrence of one string within another, working from the end of the string.
Int	Returns the integer portion of a number.
VBScript IsArray Function	Returns a Boolean value indicating whether a variable is an array.
VBScript IsDate Function	Returns a Boolean value indicating whether an expression can be converted to a date.
VBScript IsEmpty Function	Returns a Boolean value indicating whether a variable has been initialized.
VBScript IsNull Function	Returns a Boolean value indicating whether an expression contains no valid data (**Null**).
VBScript IsNumeric Function	Returns a Boolean value indicating whether an expression can be evaluated as a number.
VBScript IsObject Function	Returns a Boolean value indicating whether an expression references a valid object.
VBScript Join Function	Returns a string created by joining a number of substrings contained in an array.
VBScript LBound Function	Returns the smallest available subscript for the indicated dimension of an array.

VBScript LCase Function	Returns a string that has been converted to lower case.
VBScript Left Function	Returns a specified number of characters from the left side of a string.
VBScript Len Function	Returns the number of characters in a string or the number of bytes required to store a variable.
VBScript LoadPicture Function	Returns a picture object.
VBScript Log Function	Returns the natural logarithm of a number.
VBScript LTrim, RTrim, Trim Function	Returns a copy of a string without leading spaces.
VBScript Mid Function	Returns a specified number of characters from a string.
VBScript Minute Function	Returns the minute of the hour (0-59)
VBScript Month Function	Returns the number of the month (1-12)
VBScript MonthName Function	Returns a string with the name of the specified month.
VBScript MsgBox Function	Displays a message in a dialog box. Client-side only.
VBScript Now Function	Returns the current date and time set on your computer system.
VBScript Oct Function	Returns a string representing the octal value of a number.
VBScript Replace Function	Returns a string in which a specified substring has been replaced with another substring a specified number of times.
VBScript RGB Function	Returns a whole number representing an RGB color value.
VBScript Right Function	Returns a specified number of characters from the right side of a string.
VBScript Rnd Function	Returns a random number.
VBScript Round Function	Returns a number rounded to a specified number of decimal places.
Rtrim	Returns a copy of a string without trailing spaces.
VBScript ScriptEngine Function	Returns a string containing the scripting language in use.
VBScript ScriptEngineBuildVersion Function	Returns the build version number of the script engine in use.
VBScript ScriptEngineMajorVersion Function	Returns the major version number of the script engine in use.
VBScript ScriptEngineMinorVersion Function	Returns the minor version number of the script engine in use.
VBScript Second Function	Returns the second (0-59).

VBScript Sgn Function	Returns an integer indicating the sign of a number.
VBScript Sin Function	Returns the sine of an angle.
VBScript Space Function	Returns a string consisting of the specified number of spaces.
VBScript Split Function	Returns an array containing a specified number of substrings.
VBScript Sqr Function	Returns the square root of a number.
VBScript StrComp Function	Returns a value indicating the result of a string comparison.
VBScript StrReverse Function	Returns a string in which the character order of a specified string is reversed.
VBScript String Function	Returns a repeating character string of the length specified.
VBScript Tan Function	Returns the tangent of an angle.
VBScript Time Function	Returns a **Date** variant with the current system time.
VBScript TimeSerial Function	Returns a **Date** variant with the time for a specific hour, minute, and second.
VBScript TimeValue Function	Returns a **Date** variant containing the time.
Trim	Returns a copy of a string without leading or trailing spaces.
VBScript TypeName Function	Returns a string that provides **Variant** subtype information about a variable.
VBScript UBound Function	Returns the largest available subscript for the indicated dimension of an array.
VBScript UCase Function	Returns a string that has been converted to upper case.
VBScript VarType Function	Returns a value indicating the subtype of a variable.
VBScript Weekday Function	Returns a whole number representing the day of the week.
VBScript WeekdayName Function	Returns a string containing the specific day of the week.
VBScript Year Function	Returns a whole number containing the year.

VBScript Abs Function

Returns the absolute value of a number.

Syntax: VBScript Abs Function

```
Abs(number)
```

Arguments: VBScript Abs Function

number

Any valid numeric expression. If *number* contains **Null**, **Null** is returned; if it is an uninitialized variable, zero is returned.

Remarks: VBScript Abs Function

The absolute value of a number is its unsigned magnitude. For example, **Abs**(-1) and **Abs**(1) both return 1.

The following example uses the **Abs** function to compute the absolute value of a number:

```
Dim MyNumber

MyNumber = Abs(50.3)   ' Returns 50.3.

MyNumber = Abs(-50.3)  ' Returns 50.3.
```

VBScript Array Function

Returns a **Variant** containing an array.

Syntax: VBScript Array Function

```
Array(arglist)
```

Arguments: VBScript Array Function

arglist

Required comma-delimited list of values that are assigned to the elements of an array contained with the **Variant**. If no argument are specified, an array of zero length is created.

Remarks: VBScript Array Function

The notation used to refer to an element of an array consists of the variable name followed by parentheses containing an index number indicating the desired element. In the following example, the first statement creates a variable named A. The second statement assigns an array to variable A. The last statement assigns the value contained in the second array element to another variable.

```
Dim A

A = Array(10,20,20)

B = A(2)
```

Note

A variable that is not declared as an array can still contain an array. Although a **Variant** variable containing an array is conceptually different from an array variable containing **Variant** elements, the array elements are accessed in the same way.

VBScript Asc Function

Returns the ANSI character code corresponding to the first letter in a string.

Syntax: VBScript Asc Function

```
Asc(string)
```

Arguments: VBScript Asc Function

string

Any valid string expression. If the *string* contains no characters, a run-time error occurs.

Remarks: VBScript Asc Function

The **AscB** function is used with byte data contained in a string. Instead of returning the character code for the first character, **AscB** returns the first byte. **AscW** is provided for 32-bit platforms that use Unicode characters. It returns the Unicode (wide) character code, thereby avoiding the conversion from Unicode to ANSI.

In the following example, **Asc** returns the ANSI character code of the first letter of each string:

```
Dim MyNumber
MyNumber = Asc("A")               ' Returns 65.
MyNumber = Asc("a")               ' Returns 97.
MyNumber = Asc("Apple")       ' Returns 65.
```

Note

The behavior of the **AscB** function depends on the byte ordering of the hardware platform, and the number of bytes used to represent Unicode characters in the system software. The function will produce different results on each supported operating system. Use with caution. The described behavior pertains to the Win32 version.

VBScript Atn Function

Returns the arctangent of a number.

Syntax: VBScript Atn Function

Atn(*number*)

Arguments: VBScript Atn Function

number

Any valid numeric expression.

Remarks: VBScript Atn Function

The **Atn** function takes the ratio of two sides of a right triangle and returns the corresponding angle in radians. The ratio is the length of the side opposite the angle divided by the length of the side adjacent to the angle.

The range of the result is -pi/2 to pi/2 radians.

To convert degrees to radians, multiply degrees by pi/180. To convert radians to degrees, multiply radians by 180/pi.

The following example uses **Atn** to calculate the value of pi:

```
Dim pi

pi = 4 * Atn(1)       ' Calculate the value of pi.
```

Note

> **Atn** is the inverse trigonometric function of **Tan**, which takes an angle as its argument and returns the ratio of two sides of a right triangle. Do not confuse **Atn** with the cotangent, which is the simple inverse of a tangent (1/tangent).

VBScript CBool Function

Returns an expression that has been converted to a **Variant** of subtype **Boolean**.

Syntax: VBScript CBool Function

```
CBool(expression)
```

Arguments: VBScript CBool Function

expression

Any valid expression.

Remarks: VBScript CBool Function

If *expression* is zero, **False** is returned; otherwise, **True** is returned. If *expression* can't be interpreted as a numeric value, a run-time error occurs.

The following example uses the **CBool** function to convert an expression to a **Boolean**. If the expression evaluates to a nonzero value, **CBool** returns **True**; otherwise, it returns **False**.

```
Dim A, B, Check

A = 5: B = 5                        ' Initialize variables.

Check = CBool(A = B)       ' Check contains True.

A = 0                                        ' Define variable.

Check = CBool(A)               ' Check contains False.
```

VBScript CByte Function

Returns an expression that has been converted to a **Variant** of subtype **Byte**.

Syntax: VBScript CByte Function

```
CByte(expression)
```

Arguments: VBScript CByte Function

expression

Any valid expression.

Remarks: VBScript CByte Function

In general, you can document your code using the subtype conversion functions to show that the result of some operation should be expressed as a particular data type rather than the default data type. For example, use **CByte** to force byte arithmetic in cases where currency, single-precision, double-precision, or integer arithmetic normally would occur.

Use the **CByte** function to provide internationally aware conversions from any other data type to a **Byte** subtype. For example, different decimal separators are properly recognized depending on the locale setting of your system, as are different thousand separators.

If *expression* lies outside the acceptable range for the **Byte** subtype, an error occurs.

The following example uses the **CByte** function to convert an expression to a byte:

```
Dim MyDouble, MyByte

MyDouble = 125.5678                 ' MyDouble is a Double.

MyByte = CByte(MyDouble)       ' MyByte contains 126.
```

VBScript CCur Function

Returns an expression that has been converted to a **Variant** of subtype **Currency**.

Syntax: VBScript CCur Function

```
CCur(expression)
```

Arguments: VBScript CCur Function

expression

Any valid expression.

Remarks: VBScript CCur Function

In general, you can document your code using the subtype conversion functions to show that the result of some operation should be expressed as a particular data type rather than the default data type. For example, use **CCur** to force currency arithmetic in cases where integer arithmetic normally would occur.

You should use the **CCur** function to provide internationally aware conversions from any other data type to a **Currency** subtype. For example, different decimal separators and thousands separators are properly recognized depending on the locale setting of your system.

The following example uses the **CCur** function to convert an expression to a Currency:

```
Dim MyDouble, MyCurr

MyDouble = 543.214588                    ' MyDouble is a Double.

MyCurr = CCur(MyDouble * 2)       ' Convert result of MyDouble * 2
(1086.429176) to a Currency (1086.4292).
```

VBScript CDate Function

Returns an expression that has been converted to a **Variant** of subtype **Date**.

Syntax: VBScript CDate Function

CDate(*date***)**

Arguments: VBScript CDate Function

date

Any valid date expression.

Remarks: VBScript CDate Function

Use the **IsDate** function to determine if *date* can be converted to a date or time. **CDate** recognizes date literals and time literals as well as some numbers that fall within the range of acceptable dates. When converting a number to a date, the whole number portion is converted to a date. Any fractional part of the number is converted to a time of day, starting at midnight.

CDate recognizes date formats according to the locale setting of your system. The correct order of day, month, and year may not be determined if it is provided in a format other than one of the recognized date settings. In addition, a long date format is not recognized if it also contains the day-of-the-week string.

The following example uses the **CDate** function to convert a string to a date. In general, hard coding dates and times as strings (as shown in this example) is not recommended. Use date and time literals (such as #10/19/1962#, #4:45:23 PM#) instead.

```
MyDate = "October 19, 1962"      ' Define date.

MyShortDate = CDate(MyDate)      ' Convert to Date data type.

MyTime = "4:35:47 PM"                 ' Define time.

MyShortTime = CDate(MyTime)      ' Convert to Date data type.
```

VBScript CDbl Function

Returns an expression that has been converted to a **Variant** of subtype **Double**.

Syntax: VBScript CDbl Function

CDbl(*expression***)**

Arguments: VBScript CDbl Function

expression

Any valid expression.

Remarks: VBScript CDbl Function

In general, you can document your code using the subtype conversion functions to show that the result of some operation should be expressed as a particular data type rather than the default data type. For example, use **CDbl** or **CSng** to force double-precision or single-precision arithmetic in cases where currency or integer arithmetic normally would occur.

Use the **CDbl** function to provide internationally aware conversions from any other data type to a **Double** subtype. For example, different decimal separators and thousands separators are properly recognized depending on the locale setting of your system.

This example uses the **CDbl** function to convert an expression to a **Double**:

```
Dim MyCurr, MyDouble

MyCurr = CCur(234.456784)                              ' MyCurr is a
Currency (234.4567).

MyDouble = CDbl(MyCurr * 8.2 * 0.01)       ' Convert result to a
Double (19.2254576).
```

VBScript Chr Function

Returns the character associated with the specified ANSI character code.

Syntax: VBScript Chr Function

Chr(*charcode***)**

Arguments: VBScript Chr Function

charcode

A number that identifies a character.

Remarks: VBScript Chr Function

Numbers from 0 to 31 are the same as standard, nonprintable ASCII codes. For example, **Chr(**10**)** returns a linefeed character.

The following example uses the **Chr** function to return the character associated with the specified character code:

```
Dim MyChar

MyChar = Chr(65)       ' Returns A.
```

```
MyChar = Chr(97)      ' Returns a.
MyChar = Chr(62)      ' Returns >.
MyChar = Chr(37)      ' Returns %.
```

Notes

The **ChrB** function is used with byte data contained in a string. Instead of returning a character, which may be one or two bytes, **ChrB** always returns a single byte. **ChrW** is provided for 32-bit platforms that use Unicode characters. Its argument is a Unicode (wide) character code, thereby avoiding the conversion from ANSI to Unicode.

The behavior of the **ChrB** function depends on the byte ordering of the hardware platform, and the number of bytes used to represent Unicode characters in the system software. The function will produce different results on each supported operating system. Use with caution. The described behavior pertains to the Win32 version.

VBScript CInt Function

Returns an expression that has been converted to a **Variant** of subtype **Integer**.

Syntax: VBScript CInt Function

CInt(*expression***)**

Arguments: VBScript CInt Function

expression

Any valid expression.

Remarks: VBScript CInt Function

In general, you can document your code using the subtype conversion functions to show that the result of some operation should be expressed as a particular data type rather than the default data type. For example, use **CInt** or **CLng** to force integer arithmetic in cases where currency, single-precision, or double-precision arithmetic normally would occur.

Use the **CInt** function to provide internationally aware conversions from any other data type to an **Integer** subtype. For example, different decimal separators are properly recognized depending on the locale setting of your system, as are different thousand separators.

If *expression* lies outside the acceptable range for the **Integer** subtype, an error occurs.

The following example uses the **CInt** function to convert a value to an Integer:

```
Dim MyDouble, MyInt
MyDouble = 2345.5678          ' MyDouble is a Double.
MyInt = CInt(MyDouble)       ' MyInt contains 2346.
```

Note

CInt differs from the **Fix** and **Int** functions, which truncate, rather than round, the fractional part of a number. When the fractional part is exactly 0.5, the **CInt** function always rounds it to the nearest even number. For example, 0.5 rounds to 0, and 1.5 rounds to 2.

VBScript CLng Function

Returns an expression that has been converted to a **Variant** of subtype **Long**.

Syntax: VBScript CLng Function

CLng(*expression***)**

Arguments: VBScript CLng Function

expression

Any valid expression.

Remarks: VBScript CLng Function

In general, you can document your code using the subtype conversion functions to show that the result of some operation should be expressed as a particular data type rather than the default data type. For example, use **CInt** or **CLng** to force integer arithmetic in cases where currency, single-precision, or double-precision arithmetic normally would occur.

Use the **CLng** function to provide internationally aware conversions from any other data type to a **Long** subtype. For example, different decimal separators are properly recognized depending on the locale setting of your system, as are different thousand separators.

If *expression* lies outside the acceptable range for the **Long** subtype, an error occurs.

The following example uses the **CLng** function to convert a value to a **Long**:

```
Dim MyVal1, MyVal2, MyLong1, MyLong2

MyVal1 = 25427.45: MyVal2 = 25427.55      ' MyVal1, MyVal2 are
Doubles.

MyLong1 = CLng(MyVal1)       ' MyLong1 contains 25427.

MyLong2 = CLng(MyVal2)       ' MyLong2 contains 25428.
```

Note

CLng differs from the **Fix** and **Int** functions, which truncate, rather than round, the fractional part of a number. When the fractional part is exactly 0.5, the **CLng** function always rounds it to the nearest even number. For example, 0.5 rounds to 0, and 1.5 rounds to 2.

VBScript Cos Function

Returns the cosine of an angle.

Syntax: VBScript Cos Function

```
Cos(number)
```

Arguments: VBScript Cos Function

number

Any valid numeric expression that expresses an angle in radians.

Remarks: VBScript Cos Function

The **Cos** function takes an angle and returns the ratio of two sides of a right triangle. The ratio is the length of the side adjacent to the angle divided by the length of the hypotenuse. The result lies in the range -1 to 1.

To convert degrees to radians, multiply degrees by pi/180. To convert radians to degrees, multiply radians by 180/pi.

The following example uses the **Cos** function to return the cosine of an angle:

```
Dim MyAngle, MySecant
MyAngle = 1.3                                   ' Define angle in
radians.
MySecant = 1 / Cos(MyAngle)       ' Calculate secant.
```

VBScript CreateObject Function

Creates and returns a reference to an Automation object. This should be used as a client-side only function. To create a server-side object, use **Server.CreateObject**. For more information, see the "Built-in Objects Reference" in this chapter.

Syntax: VBScript CreateObject Function

```
CreateObject(servername.typename)
```

Arguments: VBScript CreateObject Function

servername

The name of the application providing the object.

typename

The type or class of the object to create.

Remarks: VBScript CreateObject Function

Automation servers provide at least one type of object. For example, a word-processing application may provide an application object, a document object, and a toolbar object.

The following code returns the version number of an instance of Excel running on a remote network computer named "myserver":

```
Function GetVersion
      Dim XLApp
      Set XLApp = CreateObject("Excel.Application", "MyServer")
      GetVersion = XLApp.Version
End Function
```

Note

The **CreateObject** function will not work with the **FileSystemObject** if EnableParentPaths is False in the Sun Chili!Soft ASP registery. In this case you must use Server. CreateObject("Scripting.FileSystemObject").

To create an Automation object, assign the object returned by **CreateObject** to an object variable:

```
Dim ExcelSheet

Set ExcelSheet = CreateObject("Excel.Sheet")
```

This code starts the application creating the object (in this case, a Microsoft Excel spreadsheet). Once an object is created, you refer to it in code using the object variable you defined. In the following example, you access properties and methods of the new object using the object variable, `ExcelSheet`, and other Excel objects, including the Application object and the Cells collection. For example:

```
' Make Excel visible through the Application object.

ExcelSheet.Application.Visible = True

' Place some text in the first cell of the sheet.

ExcelSheet.Cells(1,1).Value = "This is column A, row 1"

' Save the sheet.

ExcelSheet.SaveAs "C:\DOCS\TEST.XLS"

' Close Excel with the Quit method on the Application
object.

ExcelSheet.Application.Quit

' Release the object variable.

Set ExcelSheet = Nothing
```

VBScript CSng Function

Returns an expression that has been converted to a **Variant** of subtype **Single**.

Syntax: VBScript CSng Function

CSng(*expression***)**

Arguments: VBScript CSng Function

expression

Any valid expression.

Remarks: VBScript CSng Function

In general, you can document your code using the data type conversion functions to show that the result of some operation should be expressed as a particular data type rather than the default data type. For example, use **CDbl** or **CSng** to force double-precision or single-precision arithmetic in cases where currency or integer arithmetic normally would occur.

Use the **CSng** function to provide internationally aware conversions from any other data type to a **Single** subtype. For example, different decimal separators are properly recognized depending on the locale setting of your system, as are different thousand separators.

If *expression* lies outside the acceptable range for the Single subtype, an error occurs.

The following example uses the **CSng** function to convert a value to a **Single**:

```
Dim MyDouble1, MyDouble2, MySingle1, MySingle2      ' MyDouble1,
MyDouble2 are Doubles.

MyDouble1 = 75.3421115: MyDouble2 = 75.3421555

MySingle1 = CSng(MyDouble1)       ' MySingle1 contains 75.34211.

MySingle2 = CSng(MyDouble2)       ' MySingle2 contains 75.34216.
```

VBScript CStr Function

Returns an expression that has been converted to a **Variant** of subtype **String**.

Syntax: VBScript CStr Function

CStr(*expression***)**

Arguments: VBScript CStr Function

expression

Any valid expression.

Remarks: VBScript CStr Function

In general, you can document your code using the data type conversion functions to show that the result of some operation should be expressed as a particular data type rather than the default data type. For example, use **CStr** to force the result to be expressed as a **String**.

You should use the **CStr** function instead of **Str** to provide internationally aware conversions from any other data type to a **String** subtype. For example, different decimal separators are properly recognized depending on the locale setting of your system.

The data in *expression* determines what is returned according to the following table:

If *expression* is	CStr returns
Boolean	A **String** containing **True** or **False**.
Date	A **String** containing a date in the short-date format of your system.
Null	A run-time error.
Empty	A zero-length **String** ("").
Error	A **String** containing the word Error followed by the error number.
Other numeric	A **String** containing the number.

The following example uses the **CStr** function to convert a numeric value to a **String**:

```
Dim MyDouble, MyString
MyDouble = 437.324                  ' MyDouble is a Double.
MyString = CStr(MyDouble)       ' MyString contains "437.324".
```

VBScript Date Function

Returns the current system date.

Syntax: VBScript Date Function

```
Date()
```

Remarks: VBScript Date Function

The following example uses the **Date** function to return the current system date:

```
Dim MyDate
MyDate = Date    ' MyDate contains the current system date.
```

VBScript DateAdd Function

Returns a date to which a specified time interval has been added.

Syntax: VBScript DateAdd Function

```
DateAdd(interval, number, date)
```

Arguments: VBScript DateAdd Function

interval

A string expression that is the interval you want to add. See Settings section for values. Required.

number

A numeric expression that is the number of interval you want to add. The numeric expression can either be positive, for dates in the future, or negative, for dates in the past. Required.

date

A **Variant** or literal representing the date to which interval is added. Required.

Settings: VBScript DateAdd Function

The interval argument can have the following values:

Setting	Description
yyyy	Year
q	Quarter
m	Month
y	Day of year
d	Day
w	Weekday
ww	Week of year
h	Hour
N	Minute
S	Second

Remarks: VBScript DateAdd Function

You can use the **DateAdd** function to add or subtract a specified time interval from a date. For example, you can use **DateAdd** to calculate a date 30 days from today or a time 45 minutes from now. To add days to *date,* you can use Day of Year ("y"), Day ("d"), or Weekday ("w").

The **DateAdd** function won't return an invalid date. The following example adds one month to January 31:

```
NewDate = DateAdd("m", 1, "31-Jan-95")
```

In this case, **DateAdd** returns 28-Feb-95, not 31-Feb-95. If *date* is 31-Jan-96, it returns 29-Feb-96 because 1996 is a leap year.

If the calculated date would precede the year 100, an error occurs.

If number isn't a **Long** value, it is rounded to the nearest whole number before being evaluated.

VBScript DateDiff Function

Returns the number of intervals between two dates.

Syntax: VBScript DateDiff Function

```
DateDiff(interval, date1, date2
   [,firstdayofweek[, firstweekofyear]])
```

Arguments: VBScript DateDiff Function

interval

A string expression that is the interval you want to use to calculate the differences between *date1* and *date2*. See Settings section for values. Required.

date1, date2

The two dates you want to use in the calculation. Required.

firstdayofweek

A constant that specifies the day of the week. If not specified, Sunday is assumed. See Settings section for values. Optional.

firstweekofyear

A constant that specifies the first week of the year. If not specified, the first week is assumed to be the week in which January 1 occurs. See Settings section for values. Optional.

Settings: VBScript DateDiff Function

The *interval* argument can have the following values:

Setting	Description
yyyy	Year
q	Quarter
m	Month
y	Day of year
d	Day
w	Weekday
ww	Week of year
h	Hour
n	Minute
s	Second

The *firstdayofweek* argument can have the following values:

Constant	Value	Description
vbUseSystem	0	Use National Language Support (NLS) API setting.
vbSunday	1	Sunday (default)

vbMonday	2	Monday
vbTuesday	3	Tuesday
vbWednesday	4	Wednesday
vbThursday	5	Thursday
vbFriday	6	Friday
vbSaturday	7	Saturday

The *firstweekofyear* argument can have the following values:

Constant	Value	Description
vbUseSystem	0	Use National Language Support (NLS) API setting.
vbFirstJan1	1	Start with the week in which January 1 occurs (default).
vbFirstFourDays	2	Start with the week that has at least four days in the new year.
vbFirstFullWeek	3	Start with the first full week of the new year.

Remarks: VBScript DateDiff Function

You can use the **DateDiff** function to determine how many specified time intervals exist between two dates. For example, you might use **DateDiff** to calculate the number of days between two dates, or the number of weeks between today and the end of the year.

To calculate the number of days between *date1* and *date2*, you can use either Day of year ("y") or Day ("d"). When *interval* is Weekday ("w"), **DateDiff** returns the number of weeks between the two dates. If *date1* falls on a Monday, **DateDiff** counts the number of Mondays until *date2*. It counts *date2* but not *date1*. If *interval* is Week ("ww"), however, the **DateDiff** function returns the number of calendar weeks between the two dates. It counts the number of Sundays between *date1* and *date2*. **DateDiff** counts *date2* if it falls on a Sunday; but it doesn't count *date1,* even if it does fall on a Sunday.

If *date1* refers to a later point in time than *date2,* the **DateDiff** function returns a negative number.

The *firstdayofweek* argument affects calculations that use the "w" and "ww" interval symbols.

If *date1* or *date2* is a date literal, the specified year becomes a permanent part of that date. However, if *date1* or *date2* is enclosed in quotation marks (" ") and you omit the year, the current year is inserted in your code each time the *date1* or *date2* expression is evaluated. This makes it possible to write code that can be used in different years.

When comparing December 31 to January 1 of the immediately succeeding year, **DateDiff** for Year ("yyyy") returns 1 even though only a day has elapsed.

The following example uses the **DateDiff** function to display the number of days between a given date and today:

```
Function DiffADate(theDate)
      DiffADate = "Days from today: " & DateDiff("d", Now, theDate)
End Function
```

VBScript DatePart Function

Returns the specified part of a given date.

Syntax: VBScript DatePart Function

```
DatePart(interval, date[, firstdayofweek[, firstweekofyear]])
```

Arguments: VBScript DatePart Function

interval

A string expression that is the interval of time you want to return. See Settings section for values. Required.

date

The date expression you want to evaluate. Required.

firstdayofweek

A constant that specifies the day of the week. If not specified, Sunday is assumed. See Settings section for values. Optional.

firstweekofyear

A constant that specifies the first week of the year. If not specified, the first week is assumed to be the week in which January 1 occurs. See Settings section for values. Optional.

Settings: VBScript DatePart Function

The *interval* argument can have the following values:

Setting	Description
yyyy	Year
q	Quarter
m	Month
y	Day of year
d	Day
w	Weekday
ww	Week of year
h	Hour
n	Minute

s	Second

The *firstdayofweek* argument can have the following values:

Constant	Value	Description
vbUseSystem	0	Use National Language Support (NLS) API setting.
vbSunday	1	Sunday (default)
vbMonday	2	Monday
vbTuesday	3	Tuesday
vbWednesday	4	Wednesday
vbThursday	5	Thursday
vbFriday	6	Friday
vbSaturday	7	Saturday

The *firstweekofyear* argument can have the following values:

Constant	Value	Description
vbUseSystem	0	Use National Language Support (NLS) API setting.
vbFirstJan1	1	Start with the week in which January 1 occurs (default).
vbFirstFourDays	2	Start with the week that has at least four days in the new year.
vbFirstFullWeek	3	Start with the first full week of the new year.

Remarks: VBScript DatePart Function

You can use the **DatePart** function to evaluate a date and return a specific interval of time. For example, you might use **DatePart** to calculate the day of the week or the current hour.

The *firstdayofweek* argument affects calculations that use the "w" and "ww" interval symbols.

If *date* is a date literal, the specified year becomes a permanent part of that date. However, if *date* is enclosed in quotation marks (" "), and you omit the year, the current year is inserted in your code each time the *date* expression is evaluated. This makes it possible to write code that can be used in different years.

This example takes a date and, using the **DatePart** function, displays the quarter of the year in which it occurs.

```
Function GetQuarter(TheDate)
      GetQuarter = DatePart("q", TheDate)
End Function
```

VBScript DateSerial Function

Returns a **Variant** of subtype **Date** for a specified year, month, and day.

Syntax: VBScript DateSerial Function

```
DateSerial(year, month, day)
```

Arguments: VBScript DateSerial Function

year

A number between 100 and 9999, inclusive, or a numeric expression.

month

Any numeric expression.

day

Any numeric expression.

Remarks: VBScript DateSerial Function

To specify a date, such as December 31, 1991, the range of numbers for each **DateSerial** argument should be in the accepted range for the unit; that is, 1-31 for days and 1-12 for months. However, you can also specify relative dates for each argument using any numeric expression that represents some number of days, months, or years before or after a certain date.

The following example uses numeric expressions instead of absolute date numbers. Here the **DateSerial** function returns a date that is the day before the first day (1 - 1) of two months before August (8 - 2) of 10 years before 1990 (1990 - 10); in other words, May 31, 1980.

```
DateSerial(1990 - 10, 8 - 2, 1 - 1)
```

For the *year* argument, values between 0 and 99, inclusive, are interpreted as the years 1900-1999. For all other *year* arguments, use a complete four-digit year (for example, 1800).

When any argument exceeds the accepted range for that argument, it increments to the next larger unit as appropriate. For example, if you specify 35 days, it is evaluated as one month and some number of days, depending on where in the year it is applied. However, if any single argument is outside the range -32,768 to 32,767, or if the date specified by the three arguments, either directly or by expression, falls outside the acceptable range of dates, an error occurs.

VBScript DateValue Function

Returns a **Variant** of subtype **Date**.

Syntax: VBScript DateValue Function

```
DateValue (date)
```

Arguments: VBScript DateValue Function

date

Normally a string expression representing a date from January 1, 100 through December 31, 9999. However, *date* can also be any expression that can represent a date, a time, or both a date and time, in that range.

Remarks: VBScript DateValue Function

If the *date* argument includes time information, **DateValue** doesn't return it. However, if *date* includes invalid time information (such as "89:98"), an error occurs.

If *date* is a string that includes only numbers separated by valid date separators, **DateValue** recognizes the order for month, day, and year according to the short date format you specified for your system. **DateValue** also recognizes unambiguous dates that contain month names, either in long or abbreviated form. For example, in addition to recognizing 12/30/1991 and 12/30/91, **DateValue** also recognizes December 30, 1991 and Dec 30, 1991.

If the year part of *date* is omitted, **DateValue** uses the current year from your computer's system date.

The following example uses the **DateValue** function to convert a string to a date. You can also use date literals to directly assign a date to a **Variant** variable, for example, MyDate = #9/11/63#.

```
Dim MyDate
MyDate = DateValue("September 11, 1963")      ' Return a date.
```

VBScript Day Function

Returns a whole number between 1 and 31, inclusive, representing the day of the month.

Syntax: VBScript Day Function

```
Day(date)
```

Arguments: VBScript Day Function

date

Any expression that can represent a date. If *date* contains **Null**, **Null** is returned.

The following example uses the **Day** function to obtain the day of the month from a specified date:

```
Dim MyDay
MyDay = Day("October 19, 1962")      ' MyDay contains 19.
```

VBScript Exp Function

Returns *e* (the base of natural logarithms) raised to a power.

Syntax: VBScript Exp Function

```
Exp(number)
```

Arguments: VBScript Exp Function

number

Any valid numeric expression.

Remarks: VBScript Exp Function

If the value of *number* exceeds 709.782712893, an error occurs. The constant *e* is approximately 2.718282.

The following example uses the **Exp** function to return *e* raised to a power:

```
Dim MyAngle, MyHSin      ' Define angle in radians.
MyAngle = 1.3      ' Calculate hyperbolic sine.
MyHSin = (Exp(MyAngle) - Exp(-1 * MyAngle)) / 2
```

Note

The **Exp** function complements the action of the **Log** function and is sometimes referred to as the antilogarithm.

VBScript Filter Function

Returns a zero-based array containing subset of a string array based on a specified filter criteria.

Syntax: VBScript Filter Function

```
Filter(InputStrings, Value[, Include[, Compare]])
```

Arguments: VBScript Filter Function

InputStrings

A one-dimensional array of strings to be searched. Required.

Value

The string to search for. Required.

Include

A Boolean value indicating whether to return substrings that include or exclude *Value*. If *Include* is **True**, **Filter** returns the subset of the array that contains *Value* as a substring. If Include is **False**, **Filter** returns the subset of the array that does not contain *Value* as a substring. Optional.

Compare

A numeric value indicating the kind of string comparison to use. See Settings section for values. Optional.

Settings: VBScript Filter Function

The *Compare* argument can have the following values:

Constant	Value	Description
vbBinaryCompare	0	Perform a binary comparison.
vbTextCompare	1	Perform a textual comparison.

Remarks: VBScript Filter Function

If no matches of *Value* are found within *InputStrings,* **Filter** returns an empty array. An error occurs if *InputStrings* is **Null** or is not a one-dimensional array. The array returned by the **Filter** function contains only enough elements to contain the number of matched items.

The following example uses the **Filter** function to return the array containing the search criteria "Mon":

```
Dim MyIndex

Dim MyArray (3)

MyArray(0) = "Sunday"

MyArray(1) = "Monday"

MyArray(2) = "Tuesday"

MyIndex = Filter(MyArray, "Mon") ' MyIndex(0) contains "Monday".
```

VBScript Fix, Int Functions

Returns the integer portion of a number.

Syntax: VBScript Fix, Int Functions

```
Int(number)

Fix(number)
```

Arguments: VBScript Fix, Int Functions

number

Any valid numeric expression. If *number* contains **Null**, **Null** is returned.

Remarks: VBScript Fix, Int Functions

Both **Int** and **Fix** remove the fractional part of *number* and return the resulting integer value.

The difference between **Int** and **Fix** is that if *number* is negative, **Int** returns the first negative integer less than or equal to *number,* whereas **Fix** returns the first negative integer greater than or equal to *number.* For example, **Int** converts -8.4 to -9, and **Fix** converts -8.4 to -8.

Fix(*number*) is equivalent to:

```
Sgn(number) * Int(Abs(number)Int)
```

The following examples illustrate how the **Int** and **Fix** functions return integer portions of numbers:

```
MyNumber = Int(99.8)         ' Returns 99.
MyNumber = Fix(99.2)         ' Returns 99.
MyNumber = Int(-99.8)       ' Returns -100.
MyNumber = Fix(-99.8)       ' Returns -99.
MyNumber = Int(-99.2)       ' Returns -100.
MyNumber = Fix(-99.2)       ' Returns -99.
```

VBScript FormatCurrency Function

Returns an expression formatted as a currency value using the currency symbol defined in the system control panel.

Syntax: VBScript FormatCurrency Function

FormatCurrency(*Expression*[,*NumDigitsAfterDecimal* [,*IncludeLeadingDigit* [,*UseParensForNegativeNumbers* [,*GroupDigits*]]]])

Arguments: VBScript FormatCurrency Function

Expression

The expression to be formatted. Required.

NumDigitsAfterDecimal

A numeric value indicating how many places to the right of the decimal are displayed. Default value is -1, which indicates that the computer's regional settings are used. Optional.

IncludeLeadingDigit

A tristate constant that indicates whether or not a leading zero is displayed for fractional values. See Settings section for values. Optional.

UseParensForNegativeNumbers

A tristate constant that indicates whether or not to place negative values within parentheses. See Settings section for values. Optional.

GroupDigits

A tristate constant that indicates whether or not numbers are grouped using the group delimiter specified in the computer's regional settings. See Settings section for values. Optional.

Settings: VBScript FormatCurrency Function

The *IncludeLeadingDigit, UseParensForNegativeNumbers,* and *GroupDigits* argument have the following settings:

Constant	Value	Description
TristateTrue	-1	True

TristateFalse	0	False
TristateUseDefault	-2	Use the setting from the computer's regional settings.

Remarks: VBScript FormatCurrency Function

When one or more optional arguments are omitted, values for omitted arguments are provided by the computer's regional settings.

The position of the currency symbol relative to the currency value is determined by the system's regional settings.

The following example uses the **FormatCurrency** function to format the expression as a currency and assign it to MyCurrency:

```
Dim MyCurrency

MyCurrency = FormatCurrency(1000)       ' MyCurrency contains
$1000.00.
```

Note

On Windows systems, all settings information comes from the Regional Settings Currency tab, except leading zero, which comes from the Number tab.

VBScript FormatDateTime Function

Returns an expression formatted as a date or time.

Syntax: VBScript FormatDateTime Function

```
FormatDateTime(Date[,NamedFormat])
```

Arguments: VBScript FormatDateTime Function

Date

The date expression to be formatted. Required.

NamedFormat

A numeric value that indicates the date/time format used. If omitted, **vbGeneralDate** is used. Optional.

Settings: VBScript FormatDateTime Function

The *NamedFormat* argument has the following settings:

Constant	Value	Description
vbGeneralDate	0	Display a date and/or time. If there is a date part, display it as a short date. If there is a time part, display it as a long time. If present, both parts are displayed.

vbLongDate	1	Display a date using the long date format specified in your computer's regional settings.
vbShortDate	2	Display a date using the short date format specified in your computer's regional settings.
vbLongTime	3	Display a time using the time format specified in your computer's regional settings.
vbShortTime	4	Display a time using the 24-hour format (hh:mm).

Remarks: VBScript FormatDateTime Function

The following example uses the FormatDateTime function to format the expression as a long date and assign it to MyDateTime:

```
Function GetCurrentDate

      ' FormatDateTime formats Date in long date.

      GetCurrentDate = FormatDateTime(Date, 1)

End Function
```

VBScript FormatNumber Function

Returns an expression formatted as a number.

Syntax: VBScript FormatNumber Function

```
FormatNumber(Expression[,NumDigitsAfterDecimal [,IncludeLeadingDigit
[,UseParensForNegativeNumbers [,GroupDigits]]]])
```

Arguments: VBScript FormatNumber Function

Expression

The expression to be formatted. Required.

NumDigitsAfterDecimal

A numeric value indicating how many places to the right of the decimal are displayed. Default value is -1, which indicates that the computer's regional settings are used. Optional.

IncludeLeadingDigit

A tristate constant that indicates whether or not a leading zero is displayed for fractional values. See Settings section for values. Optional.

UseParensForNegativeNumbers

A tristate constant that indicates whether or not to place negative values within parentheses. See Settings section for values. Optional.

GroupDigits

A tristate constant that indicates whether or not numbers are grouped using the group delimiter specified in the control panel. See Settings section for values. Optional.

Settings: VBScript FormatNumber Function

The *IncludeLeadingDigit, UseParensForNegativeNumbers,* and *GroupDigits* arguments have the following settings:

Constant	Value	Description
TristateTrue	-1	True
TristateFalse	0	False
TristateUseDefault	-2	Use the setting from the computer's regional settings.

Remarks: VBScript FormatNumber Function

When one or more of the optional arguments are omitted, the values for omitted arguments are provided by the computer's regional settings.

The following example uses the **FormatNumber** function to format a number to have four decimal places:

```
Function FormatNumberDemo

      Dim MyAngle, MySecant, MyNumber

      MyAngle = 1.3      ' Define angle in radians.

      MySecant = 1 / Cos(MyAngle)      ' Calculate secant.

      FormatNumberDemo = FormatNumber(MySecant,4) ' Format MySecant
to four decimal places.

End Function
```

Note

On Windows systems, all settings information comes from the Regional Settings Number tab.

VBScript FormatPercent Function

Returns an expression formatted as a percentage (multiplied by 100) with a trailing % character.

Syntax: VBScript FormatPercent Function

```
FormatPercent(Expression[,NumDigitsAfterDecimal
[,IncludeLeadingDigit [,UseParensForNegativeNumbers
[,GroupDigits]]]])
```

Arguments: VBScript FormatPercent Function

Expression

The expression to be formatted. Required.

NumDigitsAfterDecimal

A numeric value indicating how many places to the right of the decimal are displayed. Default value is -1, which indicates that the computer's regional settings are used. Optional.

IncludeLeadingDigit

The tristate constant that indicates whether or not a leading zero is displayed for fractional values. See Settings section for values. Optional.

UseParensForNegativeNumbers

The tristate constant that indicates whether or not to place negative values within parentheses. See Settings section for values. Optional.

GroupDigits

The tristate constant that indicates whether or not numbers are grouped using the group delimiter specified in the control panel. See Settings section for values. Optional.

Settings: VBScript FormatPercent Function

The *IncludeLeadingDigit, UseParensForNegativeNumbers,* and *GroupDigits* arguments have the following settings:

Constant	Value	Description
TristateTrue	-1	True
TristateFalse	0	False
TristateUseDefault	-2	Use the setting from the computer's regional settings.

Remarks: VBScript FormatPercent Function

When one or more optional arguments are omitted, the values for the omitted arguments are provided by the computer's regional settings.

The following example uses the **FormatPercent** function to format an expression as a percent:

```
Dim MyPercent

MyPercent = FormatPercent(2/32) ' MyPercent contains 6.25%.
```

Note

On Windows systems, all settings information comes from the Regional Settings Number tab.

VBScript GetObject Function

Returns a reference to an Automation object from a file. This feature is for Windows only.

Syntax: VBScript GetObject Function

```
GetObject([pathname] [, appname.objectype])
```

Arguments: VBScript GetObject Function

pathname

An optional string that is the full path and name of the file containing the object to retrieve. If *pathname* is omitted, *class* is required.

appname

A required string. Name of the application providing the object.

objectype

A required string. Type or class of object to create.

Remarks: VBScript GetObject Function

Use the **GetObject** function to access an Automation object from a file and assign the object to an object variable. Use the **Set** statement to assign the object returned by **GetObject** to the object variable. For example:

```
Dim CADObject

Set CADObject = GetObject("C:\CAD\SCHEMA.CAD")
```

When this code is executed, the application associated with the specified pathname is started and the object in the specified file is activated. If *pathname* is a zero-length string (""), **GetObject** returns a new object instance of the specified type. If the *pathname* argument is omitted, **GetObject** returns a currently active object of the specified type. If no object of the specified type exists, an error occurs.

Some applications allow you to activate part of a file. Add an exclamation point (!) to the end of the file name and follow it with a string that identifies the part of the file you want to activate. For information on how to create this string, see the documentation for the application that created the object.

For example, in a drawing application you might have multiple layers to a drawing stored in a file. You could use the following code to activate a layer within a drawing called SCHEMA.CAD:

```
Set LayerObject = GetObject("C:\CAD\SCHEMA.CAD!Layer3")
```

If you don't specify the object's class, Automation determines the application to start and the object to activate, based on the file name you provide. Some files, however, may support more than one class of object. For example, a drawing might support three different types of objects: an Application object, a Drawing object, and a Toolbar object, all of which are part of the same file. To specify which object in a file you want to activate, use the optional appname.objectype arguments. For example:

```
Dim MyObject
```

```
Set MyObject = GetObject("C:\DRAWINGS\SAMPLE.DRW",
"FIGMENT.DRAWING")
```

In the preceding example, `FIGMENT` is the name of a drawing application and `DRAWING` is one of the object types it supports. Once an object is activated, you reference it in code using the object variable you defined. In the preceding example, you access properties and methods of the new object using the object variable `MyObject.` For example:

```
MyObject.Line 9, 90

MyObject.InsertText 9, 100, "Hello, world."

MyObject.SaveAs "C:\DRAWINGS\SAMPLE.DRW"
```

Note

Use the **GetObject** function when there is a current instance of the object or if you want to create the object with a file already loaded. If there is no current instance, and you don't want the object started with a file loaded, use the **CreateObject** function.

If an object has registered itself as a single-instance object, only one instance of the object is created, no matter how many times **CreateObject** is executed. With a single-instance object, **GetObject** always returns the same instance when called with the zero-length string ("") syntax, and it causes an error if the *pathname* argument is omitted.

VBScript Hex Function

Returns a string representing the hexadecimal value of a number.

Syntax: VBScript Hex Function

```
Hex(number)
```

Arguments: VBScript Hex Function

number

Any valid expression.

Remarks: VBScript Hex Function

If *number* is not already a whole number, it is rounded to the nearest whole number before being evaluated.

If *number* is	Hex returns
Null	Null.
Empty	Zero (0).
Any other number	Up to eight hexadecimal characters.

You can represent hexadecimal numbers directly by preceding numbers in the proper range with &H. For example, &H10 represents decimal 16 in hexadecimal notation.

The following example uses the **Hex** function to return the hexadecimal value of a number:

```
Dim MyHex

MyHex = Hex(5)       ' Returns 5.

MyHex = Hex(10)       ' Returns A.

MyHex = Hex(459)       ' Returns 1CB.
```

VBScript Hour Function

Returns a whole number between 0 and 23, inclusive, representing the hour of the day.

Syntax: VBScript Hour Function

```
Hour(time)
```

Arguments: VBScript Hour Function

time

Any expression that can represent a time. If *time* contains **Null**, **Null** is returned.

Remarks: VBScript Hour Function

The following example uses the **Hour** function to obtain the hour from the current time:

```
Dim MyTime, MyHour

MyTime = Now

MyHour = Hour(MyTime) ' MyHour contains the number representing
                                                ' the current hour.
```

VBScript InputBox Function

Displays a prompt in a dialog box, waits for the user to input text or click a button, and returns the contents of the text box. *This is a client-side only function*.

Syntax: VBScript InputBox Function

```
InputBox(prompt[, title][, default][, xpos][, ypos][, helpfile,
context])
```

Arguments: VBScript InputBox Function

prompt

A string expression displayed as the message in the dialog box. The maximum length of *prompt* is approximately 1024 characters, depending on the width of the characters used. If *prompt* consists of more than one line, you can separate the lines using a carriage return character (**Chr**(13)), a linefeed character (**Chr**(10)), or carriage return-linefeed character combination (**Chr**(13) **&** **Chr**(10)) between each line.

title

A string expression displayed in the title bar of the dialog box. If you omit *title,* the application name is placed in the title bar.

default

A string expression displayed in the text box as the default response if no other input is provided. If you omit *default,* the text box is displayed empty.

xpos

A numeric expression that specifies, in twips, the horizontal distance of the left edge of the dialog box from the left edge of the screen. If *xpos* is omitted, the dialog box is horizontally centered.

ypos

A numeric expression that specifies, in twips, the vertical distance of the upper edge of the dialog box from the top of the screen. If *ypos* is omitted, the dialog box is vertically positioned approximately one-third of the way down the screen.

helpfile

A string expression that identifies the Help file to use to provide context-sensitive Help for the dialog box. If *helpfile* is provided, *context* must also be provided.

context

A numeric expression that identifies the Help context number assigned by the Help author to the appropriate Help topic. If *context* is provided, *helpfile* must also be provided.

Remarks: VBScript InputBox Function

When both *helpfile* and *context* are supplied, a Help button is automatically added to the dialog box.

If the user clicks **OK** or presses **ENTER**, the **InputBox** function returns whatever is in the text box. If the user clicks **Cancel**, the function returns a zero-length string ("").

The following example uses the **InputBox** function to display an input box and assign the string to the variable Input:

```
Dim Input

Input = InputBox("Enter your name")

MsgBox ("You entered: " & Input)
```

VBScript InStr Function

Returns the position of the first occurrence of one string within another.

Syntax: VBScript InStr Function

```
InStr([start, ]string1, string2[, compare])
```

Arguments: VBScript InStr Function

start

The numeric expression that sets the starting position for each search. If omitted, search begins at the first character position. If *start* contains **Null**, an error occurs. The *start* argument is required if *compare* is specified. Optional.

string1

The string expression being searched. Required.

string2

The string expression searched for. Required.

compare

A numeric value indicating the kind of comparison to use when evaluating substrings. See Settings section for values. If omitted, a binary comparison is performed. Optional.

Settings : VBScript InStr Function

The *compare* argument can have the following values:

Constant	Value	Description
vbBinaryCompare	0	Perform a binary comparison.
vbTextCompare	1	Perform a textual comparison.

Return Values: VBScript InStr Function

The **InStr** function returns the following values:

If	InStr returns
string1 is zero-length	0
string1 is Null	**Null**
string2 is zero-length	Start
***string2* is Null**	Null
string2 is not found	0
string2 is found within string1	Position at which match is found
start > **Len**(*string2*)	0

Remarks: VBScript InStr Function

The following examples use **InStr** to search a string:

```
Dim SearchString, SearchChar, MyPos

SearchString ="XXpXXpXXPXXP"      ' String to search in.

SearchChar = "P"      ' Search for "P".
```

```
MyPos = Instr(4, SearchString, SearchChar, 1)      ' A textual
comparison starting at position 4. Returns 6.

MyPos = Instr(1, SearchString, SearchChar, 0)      ' A binary
comparison starting at position 1. Returns 9.

MyPos = Instr(SearchString, SearchChar)      ' Comparison is binary
by default (last argument is omitted). Returns 9.

MyPos = Instr(1, SearchString, "W")      ' A binary comparison
starting at position 1. Returns 0 ("W" is not found).
```

Note

The **InStrB** function is used with byte data contained in a string. Instead of returning the character position of the first occurrence of one string within another, **InStrB** returns the byte position.

VBScript InStrRev Function

Returns the position of an occurrence of one string within another, from the end of string.

Syntax: VBScript InStrRev Function

```
InStrRev(string1, string2[, start[, compare]])
```

Arguments: VBScript InStrRev Function

string1

The string expression being searched. Required.

string2

The string expression being searched for. Required.

start

The numeric expression that sets the starting position for each search. If omitted, -1 is used, which means that the search begins at the last character position. If *start* contains **Null**, an error occurs. Optional.

compare

The numeric value indicating the kind of comparison to use when evaluating substrings. If omitted, a binary comparison is performed. See Settings section for values. Optional.

Settings: VBScript InStrRev Function

The *compare* argument can have the following values:

Constant	Value	Description
vbBinaryCompare	0	Perform a binary comparison.
vbTextCompare	1	Perform a textual comparison.

Return Values: VBScript InStrRev Function

InStrRev returns the following values:

If	InStrRev returns
string1 is zero-length	0
string1 is **Null**	Null
string2 is zero-length	Start
string2 is **Null**	Null
string2 is not found	0
string2 is found within string1	Position at which match is found
start > **Len**(string2)	0

Remarks: VBScript InStrRev Function

Note that the syntax for the **InStrRev** function is not the same as the syntax for the **InStr** function.

The following examples use the **InStrRev** function to search a string:

```
Dim SearchString, SearchChar, MyPos

SearchString ="XXpXXpXXPXXP"      ' String to search in.

SearchChar = "P"      ' Search for "P".

MyPos = InstrRev(SearchString, SearchChar, 10, 0)      ' A binary
comparison starting at position 10. Returns 9.

MyPos = InstrRev(SearchString, SearchChar, -1, 1)      ' A textual
comparison starting at the last position. Returns 12.

MyPos = InstrRev(SearchString, SearchChar, 8)      ' Comparison is
binary by default (last argument is omitted). Returns 0.
```

VBScript IsArray Function

Returns a Boolean value indicating whether a variable is an array.

Syntax: VBScript IsArray Function

```
IsArray (varname)
```

Arguments: VBScript IsArray Function

varname

Any variable.

Remarks: VBScript IsArray Function

IsArray returns **True** if the variable is an array; otherwise, it returns **False**. **IsArray** is especially useful with variants containing arrays.

The following example uses the **IsArray** function to test whether `MyVariable` is an array:

```
Dim MyVariable

Dim MyArray(3)

MyArray(0) = "Sunday"

MyArray(1) = "Monday"

MyArray(2) = "Tuesday"

MyVariable = IsArray(MyArray) ' MyVariable contains "True".
```

VBScript IsDate Function

Returns a Boolean value indicating whether an expression can be converted to a date.

Syntax: VBScript IsDate Function

```
IsDate(expression)
```

Arguments: VBScript IsDate Function

expression

Any date expression or string expression recognizable as a date or time.

Remarks: VBScript IsDate Function

IsDate returns **True** if the expression is a date or can be converted to a valid date; otherwise, it returns **False**. In Microsoft Windows, the range of valid dates is January 1, 100 AD through December 31, 9999 AD; the ranges vary among operating systems.

The following example uses the **IsDate** function to determine whether an expression can be converted to a date:

```
Dim MyDate, YourDate, NoDate, MyCheck

MyDate = "October 19, 1962": YourDate = #10/19/62#: NoDate = "Hello"

MyCheck = IsDate(MyDate)      ' Returns True.

MyCheck = IsDate(YourDate)      ' Returns True.

MyCheck = IsDate(NoDate)      ' Returns False.
```

VBScript IsEmpty Function

Returns a Boolean value indicating whether a variable has been initialized.

Syntax: VBScript IsEmpty Function

```
IsEmpty(expression)
```

Arguments: VBScript IsEmpty Function

expression

Any expression. However, because **IsEmpty** is used to determine if individual variables are initialized, the *expression* argument is most often a single variable name.

Remarks: VBScript IsEmpty Function

IsEmpty returns **True** if the variable is uninitialized, or is explicitly set to **Empty**; otherwise, it returns **False**. **False** is always returned if *expression* contains more than one variable.

The following example uses the **IsEmpty** function to determine whether a variable has been initialized:

```
Dim MyVar, MyCheck

MyCheck = IsEmpty(MyVar)      ' Returns True.

MyVar = Null      ' Assign Null.

MyCheck = IsEmpty(MyVar)      ' Returns False.

MyVar = Empty      ' Assign Empty.

MyCheck = IsEmpty(MyVar)      ' Returns True.
```

VBScript IsNull Function

Returns a Boolean value that indicates whether an expression contains no valid data (**Null**).

Syntax: VBScript IsNull Function

```
IsNull(expression)
```

Arguments: VBScript IsNull Function

expression

Any expression.

Remarks: VBScript IsNull Function

IsNull returns **True** if *expression* is **Null**; that is, it contains no valid data; otherwise, **IsNull** returns **False**. If *expression* consists of more than one variable, **Null** in any constituent variable causes **True** to be returned for the entire expression.

The **Null** value indicates that the variable contains no valid data. **Null** is not the same as **Empty**, which indicates that a variable has not yet been initialized. It is also not the same as a zero-length string (""), which is sometimes referred to as a null string.

The following example uses the **IsNull** function to determine whether a variable contains a **Null**:

```
Dim MyVar, MyCheck

MyCheck = IsNull(MyVar)      ' Returns False.

MyVar = Null      ' Assign Null.
```

```
MyCheck = IsNull(MyVar)       ' Returns True.

MyVar = Empty      ' Assign Empty.

MyCheck = IsNull(MyVar)       ' Returns False.
```

Note

Use the **IsNull** function to determine whether an expression contains a **Null** value. Expressions that you might expect to evaluate to **True** under some circumstances, such as If Var = Null and If Var <> Null, are always **False**. This is because any expression containing a **Null** is itself **Null**, and therefore, **False**.

VBScript IsNumeric Function

Returns a Boolean value indicating whether an expression can be evaluated as a number.

Syntax: VBScript IsNumeric Function

```
IsNumeric (expression)
```

Arguments: VBScript IsNumeric Function

expression

Any expression.

Remarks: VBScript IsNumeric Function

IsNumeric returns **True** if the entire expression is recognized as a number; otherwise, it returns **False**.

IsNumeric returns **False** if *expression* is a date expression.

The following example uses the **IsNumeric** function to determine whether a variable can be evaluated as a number:

```
Dim MyVar, MyCheck

MyVar = 53      ' Assign a value.

MyCheck = IsNumeric(MyVar)       ' Returns True.

MyVar = "459.95"      ' Assign a value.

MyCheck = IsNumeric(MyVar)       ' Returns True.

MyVar = "45 Help"      ' Assign a value.

MyCheck = IsNumeric(MyVar)       ' Returns False.
```

VBScript IsObject Function

Returns a Boolean value indicating whether an expression references a valid Automation object.

Syntax: VBScript IsObject Function

```
IsObject(expression)
```

Arguments: VBScript IsObject Function

expression

Any expression.

Remarks: VBScript IsObject Function

IsObject returns **True** if *expression* is a variable of **Object** subtype or a user-defined object; otherwise, it returns **False**.

The following example uses the **IsObject** function to determine if an identifier represents an object variable:

```
Dim MyInt, MyCheck, MyObject

Set MyObject = Me

MyCheck = IsObject(MyObject)      ' Returns True.

MyCheck = IsObject(MyInt)      ' Returns False.
```

VBScript Join Function

Returns a string created by joining a number of substrings contained in an array.

Syntax: VBScript Join Function

```
Join(list[, delimiter])
```

Arguments: VBScript Join Function

list

A one-dimensional array containing substrings to be joined. Required.

delimiter

A string character used to separate the substrings in the returned string. If omitted, the space character (" ") is used. If *delimiter* is a zero-length string, all items in the list are concatenated with no delimiters. Optional.

Remarks: VBScript Join Function

The following example uses the **Join** function to join the substrings of `MyArray`:

```
Dim MyString

Dim MyArray(3)

MyArray(0) = "Mr."

MyArray(1) = "John "

MyArray(2) = "Doe "
```

```
MyArray(3) = "III"
MyString = Join(MyArray) ' MyString contains "Mr. John Doe III".
```

VBScript LCase Function

Returns a string that has been converted to lowercase.

Syntax: VBScript LCase Function

```
LCase(string)
```

Arguments: VBScript LCase Function

string

Any valid string expression. If *string* contains **Null**, **Null** is returned.

Remarks: VBScript LCase Function

Only uppercase letters are converted to lowercase; all lowercase letters and nonletter characters remain unchanged.

The following example uses the **LCase** function to convert uppercase letters to lowercase:

```
Dim MyString
Dim LCaseString
MyString = "VBSCript"
LCaseString = LCase(MyString)      ' LCaseString contains
"vbscript".
```

VBScript LBound Function

Returns the smallest available subscript for the indicated dimension of an array.

Syntax: VBScript LBound Function

```
LBound(arrayname[, dimension])
```

Arguments: VBScript LBound Function

arrayname

The name of the array variable; follows standard variable naming conventions.

dimension

A whole number indicating which dimension's lower bound is returned. Use 1 for the first dimension, 2 for the second, and so on. If *dimension* is omitted, 1 is assumed.

Remarks: VBScript LBound Function

The **LBound** function is used with the UBound Function to determine the size of an array. Use the **UBound** function to find the upper limit of an array dimension.

The default lower bound for any dimension is always 0.

VBScript Left Function

Returns a specified number of characters from the left side of a string.

Syntax: VBScript Left Function

```
Left(string, length)
```

Arguments: VBScript Left Function

string

A string expression from which the leftmost characters are returned. If *string* contains **Null**, **Null** is returned.

length

A numeric expression indicating how many characters to return. If 0, a zero-length string ("") is returned. If greater than or equal to the number of characters in *string,* the entire string is returned.

Remarks: VBScript Left Function

To determine the number of characters in *string,* use the **Len** function.

The following example uses the **Left** function to return the first three characters of `MyString`:

```
Dim MyString, LeftString

MyString = "VBSCript"

LeftString = Left(MyString, 3) ' LeftString contains "VBS".
```

Notes

The **LeftB** function is used with byte data contained in a string. Instead of specifying the number of characters to return, *length* specifies the number of bytes.

The behavior of the **LeftB** function depends on the byte ordering of the hardware platform, and the number of bytes used to represent Unicode characters in the system software. The function will produce different results on each supported operating system. Use with caution. The described behavior pertains to the Win32 version.

VBScript Len Function

Returns the number of characters in a string or the number of bytes required to store a variable.

Syntax: VBScript Len Function

```
Len(string | varname)
```

Arguments: VBScript Len Function

string

Any valid string expression. If *string* contains **Null**, **Null** is returned.

varname

Any valid variable name. If *varname* contains **Null**, **Null** is returned.

Remarks: VBScript Len Function

The following example uses the **Len** function to return the number of characters in a string:

```
Dim MyString

MyString = Len("VBSCRIPT") ' MyString contains 8.
```

Note

The **LenB** function is used with byte data contained in a string. Instead of returning the number of characters in a string, **LenB** returns the number of bytes used to represent that string.

VBScript LoadPicture Function

Returns a picture object. Available only on 32-bit Windows platforms.

Syntax: VBScript LoadPicture Function

```
LoadPicture(picturename)
```

Arguments: VBScript LoadPicture Function

picturename

A string expression that indicates the name of the picture file to be loaded.

Remarks: VBScript LoadPicture Function

Graphics formats recognized by **LoadPicture** include bitmap (.bmp) files, icon (.ico) files, run-length encoded (.rle) files, metafile (.wmf) files, enhanced metafiles (.emf), GIF (.gif) files, and JPEG (.jpg) files.

VBScript Log Function

Returns the natural logarithm of a number.

Syntax: VBScript Log Function

```
Log(number)
```

Arguments: VBScript Log Function

number

Any valid numeric expression greater than 0.

Remarks: VBScript Log Function

The natural logarithm is the logarithm to the base *e*. The constant *e* is approximately 2.718282.

You can calculate *base-n* logarithms for any number *x* by dividing the natural logarithm of *x* by the natural logarithm of *n* as follows:

```
Logn(x) = Log(x) / Log(n)
```

The following example illustrates a custom **Function** that calculates base-10 logarithms:

```
Function Log10(X)

Log10 = Log(X) / Log(10)

End Function
```

VBScript LTrim, RTrim, Trim Function

Returns a copy of a string without leading spaces (**LTrim**), trailing spaces (**RTrim**), or both leading and trailing spaces (**Trim**).

Syntax: VBScript LTrim, RTrim, Trim Function

```
LTrim(string)

RTrim(string)

Trim(string)
```

Arguments: VBScript LTrim, RTrim, Trim Function

string

Any valid string expression. If *string* contains **Null**, **Null** is returned.

Remarks: VBScript LTrim, RTrim, Trim Function

The following example uses the **LTrim**, **RTrim**, and **Trim** functions to trim leading spaces, trailing spaces, and both leading and trailing spaces, respectively:

```
Dim MyVar

MyVar = LTrim("      vbscript ")      ' MyVar contains "vbscript ".

MyVar = RTrim("      vbscript ")      ' MyVar contains "
vbscript".

MyVar = Trim("      vbscript ")      ' MyVar contains "vbscript".
```

VBScript Mid Function

Returns a specified number of characters from a string.

Syntax: VBScript Mid Function

```
Mid(string, start[, length])
```

Arguments: VBScript Mid Function

string

The string expression from which characters are returned. If *string* contains **Null**, **Null** is returned.

start

The character position in *string* at which the part to be taken begins. If *start* is greater than the number of characters in *string,* Mid returns a zero-length string ("").

length

The number of characters to return. If omitted or if there are fewer than *length* characters in the text (including the character at *start*), all characters from the *start* position to the end of the string are returned.

Remarks: VBScript Mid Function

To determine the number of characters in *string,* use the **Len** function.

The following example uses the **Mid** function to return six characters, beginning with the fourth character, in a string:

```
Dim MyVar

MyVar = Mid("VB Script is fun!", 4, 6) ' MyVar contains "Script".
```

Notes

The **MidB** function is used with byte data contained in a string. Instead of specifying the number of characters, the arguments specify numbers of bytes.

The behavior of the **MidB** function depends on the byte ordering of the hardware platform, and the number of bytes used to represent Unicode characters in the system software. The function will produce different results on each supported operating system. Use with caution. The described behavior pertains to the Win32 version.

VBScript Minute Function

Returns a whole number between 0 and 59, inclusive, representing the minute of the hour.

Syntax: VBScript Minute Function

```
Minute(time)
```

Arguments: VBScript Minute Function

time

Any expression that can represent a time. If *time* contains **Null**, **Null** is returned.

Remarks: VBScript Minute Function

The following example uses the **Minute** function to return the minute of the hour:

```
Dim MyVar

MyVar = Minute(Now)
```

VBScript Month Function

Returns a whole number between 1 and 12, inclusive, representing the month of the year.

Syntax: VBScript Month Function

```
Month(date)
```

Arguments: VBScript Month Function

date

Any expression that can represent a date. If *date* contains **Null**, **Null** is returned.

Remarks: VBScript Month Function

The following example uses the **Month** function to return the current month:

```
Dim MyVar

MyVar = Month(Now) ' MyVar contains the number corresponding to
                                        ' the current month.
```

VBScript MonthName Function

Returns a string indicating the specified month.

Syntax: VBScript MonthName Function

```
MonthName(month[, abbreviate])
```

Arguments: VBScript MonthName Function

month

The numeric designation of the month. For example, January is 1, February is 2, and so on. Required.

abbreviate

A Boolean value that indicates if the month name is to be abbreviated. If omitted, the default is **False**, which means that the month name is not abbreviated. Optional.

Remarks: VBScript MonthName Function

The following example uses the **MonthName** function to return an abbreviated month name for a date expression:

```
Dim MyVar

MyVar = MonthName(10, True) ' MyVar contains "Oct".
```

VBScript MsgBox Function

Displays a message in a dialog box, waits for the user to click a button, and returns a value indicating which button the user clicked. *This is a client-side function only.*

Syntax: VBScript MsgBox Function

```
MsgBox(prompt[, buttons][, title][, helpfile, context])
```

Arguments: VBScript MsgBox Function

prompt

A string expression displayed as the message in the dialog box. The maximum length of *prompt* is approximately 1024 characters, depending on the width of the characters used. If *prompt* consists of more than one line, you can separate the lines using a carriage return character (**Chr**(13)), a linefeed character (**Chr**(10)), or carriage return-linefeed character combination (**Chr**(13) **&** **Chr**(10)) between each line.

buttons

A numeric expression that is the sum of values specifying the number and type of buttons to display, the icon style to use, the identity of the default button, and the modality of the message box. See Settings section for values. If omitted, the default value for *buttons* is 0.

title

A string expression displayed in the title bar of the dialog box. If you omit *title,* the application name is placed in the title bar.

helpfile

A string expression that identifies the Help file to use to provide context-sensitive Help for the dialog box. If *helpfile* is provided, *context* must also be provided. Not available on 16-bit platforms.

context

A numeric expression that identifies the Help context number assigned by the Help author to the appropriate Help topic. If *context* is provided, *helpfile* must also be provided. Not available on 16-bit platforms.

Settings: VBScript MsgBox Function

The *buttons* argument settings are:

Constant	Value	Description
vbOKOnly	0	Display **OK** button only.
vbOKCancel	1	Display **OK** and **Cancel** buttons.
vbAbortRetryIgnore	2	Display **Abort, Retry,** and **Ignore** buttons.
vbYesNoCancel	3	Display **Yes, No,** and **Cancel** buttons.
vbYesNo	4	Display **Yes** and **No** buttons.
vbRetryCancel	5	Display **Retry** and **Cancel** buttons.
vbCritical	16	Display Critical Message icon.
vbQuestion	32	Display Warning Query icon.
vbExclamation	48	Display Warning Message icon.
vbInformation	64	Display Information Message icon.
vbDefaultButton1	0	First button is default.
vbDefaultButton2	256	Second button is default.
vbDefaultButton3	512	Third button is default.
vbDefaultButton4	768	Fourth button is default.
vbApplicationModal	0	Application modal; the user must respond to the message box before continuing work in the current application.
vbSystemModal	4096	System modal; all applications are suspended until the user responds to the message box.

The first group of values (0-5) describes the number and type of buttons displayed in the dialog box; the second group (16, 32, 48, 64) describes the icon style; the third group (0, 256, 512, 768) determines which button is the default; and the fourth group (0, 4096) determines the modality of the message box. When adding numbers to create a final value for the argument *buttons,* use only one number from each group.

Return Values: VBScript MsgBox Function

The **MsgBox** function has the following return values:

Constant	Value	Button
vbOK	1	OK
vbCancel	2	Cancel
vbAbort	3	Abort
vbRetry	4	Retry
vbIgnore	5	Ignore
vbYes	6	Yes

vbNo	7	No

Remarks: VBScript MsgBox Function

When both *helpfile* and *context* are provided, the user can press F1 to view the Help topic corresponding to the context.

If the dialog box displays a **Cancel** button, pressing the **ESC** key has the same effect as clicking **Cancel.** If the dialog box contains a **Help** button, context-sensitive Help is provided for the dialog box. However, no value is returned until one of the other buttons is clicked.

The following example uses the **MsgBox** function to display a message box and return a value describing which button was clicked:

```
Dim MyVar

MyVar = MsgBox ("Hello World!", 65, "MsgBox Example")

      ' MyVar contains either 1 or 2, depending on which button is
clicked.
```

VBScript Now Function

Returns the current date and time according to the setting of your computer's system date and time.

Syntax: VBScript Now Function

```
Now()
```

Remarks: VBScript Now Function

The following example uses the **Now** function to return the current date and time:

```
Dim MyVar

MyVar = Now ' MyVar contains the current date and time.
```

VBScript Oct Function

Returns a string representing the octal value of a number.

Syntax: VBScript Oct Function

```
Oct(number)
```

Arguments: VBScript Oct Function

number

Any valid expression.

Remarks: VBScript Oct Function

If *number* is not already a whole number, it is rounded to the nearest whole number before being evaluated.

If *number* is	Oct returns
Null	Null.
Empty	Zero (0).
Any other number	Up to 11 octal characters,

You can represent octal numbers directly by preceding numbers in the proper range with &O. For example, &O10 is the octal notation for decimal 8.

The following example uses the **Oct** function to return the octal value of a number:

```
Dim MyOct

MyOct = Oct(4)           ' Returns 4.

MyOct = Oct(8)           ' Returns 10.

MyOct = Oct(459)       ' Returns 713.
```

VBScript Replace Function

Returns a string in which a specified substring has been replaced with another substring a specified number of times.

Syntax: VBScript Replace Function

```
Replace(expression, find, replacewith[, start[, count[, compare]]])
```

Arguments: VBScript Replace Function

expression

A string expression containing substring to replace. Required.

find

The substring being searched for. Required.

replacewith

The replacement substring. Required.

start

The position within *expression* where substring search is to begin. If omitted, 1 is assumed. Optional.

count

The number of substring substitutions to perform. If omitted, the default value is -1, which means make all possible substitutions. Optional.

compare

The numeric value indicating the kind of comparison to use when evaluating substrings. See Settings section for values. Optional.

Settings: VBScript Replace Function

The *compare* argument can have the following values:

Constant	Value	Description
vbBinaryCompare	0	Perform a binary comparison.
vbTextCompare	1	Perform a textual comparison.

Return Values: VBScript Replace Function

Replace returns the following values:

If	Replace returns
expression is zero-length	Zero-length string ("").
expression is **Null**	An error.
find is zero-length	Copy of *expression*.
replacewith is zero-length	Copy of *expression* with all occurrences of *find* removed.
start > **Len**(*expression*)	Zero-length string.
count is 0	Copy of *expression*.

Remarks: VBScript Replace Function

The return value of the **Replace** function is a string, with substitutions made, that begins at the position specified by *start* and concludes at the end of the *expression* string. It is not a copy of the original string from start to finish.

The following example uses the **Replace** function to return a string:

```
Dim MyString

MyString = Replace("XXpXXPXXp", "p", "Y")      ' A binary comparison
starting at the beginning of the string. Returns "XXYXXPXXY".

MyString = Replace("XXpXXPXXp", "p", "Y",      ' A textual
comparison starting at position 3. Returns "YXXYXXY". 3, -1, 1)
```

VBScript RGB Function

Returns a whole number representing an RGB color value.

Syntax: VBScript RGB Function

```
RGB(red, green, blue)
```

Arguments: VBScript RGB Function

red

A number in the range 0-255 representing the red component of the color. Required.

green

A number in the range 0-255 representing the green component of the color. Required.

blue

A number in the range 0-255 representing the blue component of the color. Required.

Remarks: VBScript RGB Function

Application methods and properties that accept a color specification expect that specification to be a number representing an RGB color value. An RGB color value specifies the relative intensity of red, green, and blue to cause a specific color to be displayed.

The low-order byte contains the value for red, the middle byte contains the value for green, and the high-order byte contains the value for blue.

For applications that require the byte order to be reversed, the following function will provide the same information with the bytes reversed:

```
Function RevRGB(red, green, blue)
RevRGB= CLng(blue + (green * 256) + (red * 65536))
End Function
```

The value for any argument to the **RGB** function that exceeds 255 is assumed to be 255.

VBScript Right Function

Returns a specified number of characters from the right side of a string.

Syntax: VBScript Right Function

```
Right(string, length)
```

Arguments: VBScript Right Function

string

A string expression from which the rightmost characters are returned. If *string* contains **Null**, **Null** is returned.

length

A numeric expression indicating how many characters to return. If 0, a zero-length string is returned. If greater than or equal to the number of characters in *string,* the entire string is returned.

Remarks: VBScript Right Function

To determine the number of characters in *string,* use the **Len** function.

The following example uses the **Right** function to return a specified number of characters from the right side of a string:

```
Dim AnyString, MyStr

AnyString = "Hello World"            ' Define string.

MyStr = Right(AnyString, 1)         ' Returns "d".

MyStr = Right(AnyString, 6)         ' Returns " World".

MyStr = Right(AnyString, 20)       ' Returns "Hello World".
```

Notes

The **RightB** function is used with byte data contained in a string. Instead of specifying the number of characters to return, *length* specifies the number of bytes.

The behavior of the **RightB** function depends on the byte ordering of the hardware platform, and the number of bytes used to represent Unicode characters in the system software. The function will produce different results on each supported operating system. Use with caution. The described behavior pertains to the Win32 version.

VBScript Rnd Function

Returns a random number.

Syntax: VBScript Rnd Function

```
Rnd[(number)]
```

Arguments: VBScript Rnd Function

number

Any valid numeric expression.

Remarks: VBScript Rnd Function

The **Rnd** function returns a value less than 1 but greater than or equal to 0.

The value of number determines how **Rnd** generates a random number:

If *number* is	Rnd generates
Less than zero	The same number every time, using *number* as the seed.
Greater than zero	The next random number in the sequence.
Equal to zero	The most recently generated number.
Not supplied	The next random number in the sequence.

For any given initial seed, the same number sequence is generated because each successive call to the **Rnd** function uses the previous number as a seed for the next number in the sequence.

Before calling **Rnd**, use the **Randomize** statement without an argument to initialize the random-number generator with a seed based on the system timer.

To produce random integers in a given range, use this formula:

```
Int((upperbound - lowerbound + 1) * Rnd + lowerbound)
```

Here, *upperbound* is the highest number in the range, and *lowerbound* is the lowest number in the range.

Note

To repeat sequences of random numbers, call the **Rnd** function with a negative argument immediately before using the **Randomize** statement with a numeric arguments. Using the **Randomize** statement with the same value for *number* does not repeat the previous sequence.

VBScript Round Function

Returns a number rounded to a specified number of decimal places.

Syntax: VBScript Round Function

```
Round(expression[, numdecimalplaces])
```

Arguments: VBScript Round Function

expression

A numeric expression being rounded. Required.

numdecimalplaces

A number indicating how many places to the right of the decimal are included in the rounding. If omitted, integers are returned by the **Round** function. Optional.

Remarks: VBScript Round Function

The following example uses the **Round** function to round a number to two decimal places:

```
Dim MyVar, pi

pi = 3.14159

MyVar = Round(pi, 2) ' MyVar contains 3.14.
```

VBScript ScriptEngine Function

Returns a string representing the scripting language in use.

Syntax: VBScript ScriptEngine Function

```
ScriptEngine()
```

Return Values: VBScript ScriptEngine Function

The **ScriptEngine** function can return any of the following strings:

String	Description
VBScript	Indicates that Microsoft® Visual Basic® Scripting Edition is the current script engine.
JScript	Indicates that Microsoft JScript™ is the current script engine.
VBA	Indicates that Microsoft® Visual Basic® for Applications is the current script engine.

Remarks: VBScript ScriptEngine Function

The following example uses the **ScriptEngine** function to return a string describing the scripting language in use:

```
Function GetScriptEngineInfo
      Dim s
      s = ""      ' Build string with necessary info.
      s = ScriptEngine & " Version "
      s = s & ScriptEngineMajorVersion & "."
      s = s & ScriptEngineMinorVersion & "."
      s = s & ScriptEngineBuildVersion
      GetScriptEngineInfo = s      ' Return the results.
End Function
```

VBScript ScriptEngineBuildVersion Function

Returns the build version number of the script engine in use.

Syntax: VBScript ScriptEngineBuildVersion Function

```
ScriptEngineBuildVersion()
```

Remarks: VBScript ScriptEngineBuildVersion Function

The return value corresponds directly to the version information contained in the DLL for the scripting language in use.

VBScript ScriptEngineMajorVersion Function

Returns the major version number of the script engine in use.

Syntax: VBScript ScriptEngineMajorVersion Function

```
ScriptEngineMajorVersion()
```

Remarks: VBScript ScriptEngineMajorVersion Function

The return value corresponds directly to the version information contained in the DLL for the scripting language in use.

The following example uses the **ScriptEngineBuildVersion** function to return the build version number of the scripting engine:

```
Function GetScriptEngineInfo

      Dim s

      s = ""      ' Build string with necessary info.

      s = ScriptEngine & " Version "

      s = s & ScriptEngineMajorVersion & "."

      s = s & ScriptEngineMinorVersion & "."

      s = s & ScriptEngineBuildVersion

      GetScriptEngineInfo = s      ' Return the results.

End Function
```

VBScript ScriptEngineMinorVersion Function

Returns the minor version number of the script engine in use.

Syntax: VBScript ScriptEngineMinorVersion Function

```
ScriptEngineMinorVersion()
```

Remarks: VBScript ScriptEngineMinorVersion Function

The return value corresponds directly to the version information contained in the DLL for the scripting language in use.

The following example uses the **ScriptEngineMinorVersion** function to return the minor version number of the scripting engine:

```
Function GetScriptEngineInfo

      Dim s

      s = ""      ' Build string with necessary info.

      s = ScriptEngine & " Version "

      s = s & ScriptEngineMajorVersion & "."

      s = s & ScriptEngineMinorVersion & "."

      s = s & ScriptEngineBuildVersion

      GetScriptEngineInfo = s      ' Return the results.

End Function
```

VBScript Second Function

Returns a whole number between 0 and 59, inclusive, representing the second of the minute.

Syntax: VBScript Second Function

Second(*time***)**

Arguments: VBScript Second Function

time

Any expression that can represent a time. If *time* contains **Null**, **Null** is returned.

Remarks: VBScript Second Function

The following example uses the **Second** function to return the current second:

```
Dim MySec
MySec = Second(Now)
      ' MySec contains the number representing the current second.
```

VBScript Sgn Function

Returns an integer indicating the sign of a number.

Syntax: VBScript Sgn Function

Sgn(*number***)**

Arguments: VBScript Sgn Function

number

Any valid numeric expression.

Return Values: VBScript Sgn Function

The **Sgn** function has the following return values:

If *number* is	**Sgn returns**
Greater than zero	1
Equal to zero	0
Less than zero	-1

Remarks: VBScript Sgn Function

The sign of the *number* argument determines the return value of the **Sgn** function.

The following example uses the **Sgn** function to determine the sign of a number:

```
Dim MyVar1, MyVar2, MyVar3, MySign
```

```
MyVar1 = 12: MyVar2 = -2.4: MyVar3 = 0
MySign = Sgn(MyVar1)      ' Returns 1.
MySign = Sgn(MyVar2)      ' Returns -1.
MySign = Sgn(MyVar3)      ' Returns 0.
```

VBScript Sin Function

Returns the sine of an angle.

Syntax: VBScript Sin Function

```
Sin(number)
```

Arguments: VBScript Sin Function

number

Any valid numeric expression that expresses an angle in radians.

Remarks: VBScript Sin Function

The **Sin** function takes an angle and returns the ratio of two sides of a right triangle. The ratio is the length of the side opposite the angle divided by the length of the hypotenuse. The result lies in the range -1 to 1.

To convert degrees to radians, multiply degrees by pi/180. To convert radians to degrees, multiply radians by 180/pi.

The following example uses the **Sin** function to return the sine of an angle:

```
Dim MyAngle, MyCosecant
MyAngle = 1.3      ' Define angle in radians.
MyCosecant = 1 / Sin(MyAngle)      ' Calculate cosecant.
```

VBScript Space Function

Returns a string consisting of the specified number of spaces.

Syntax: VBScript Space Function

```
Space(number)
```

Arguments: VBScript Space Function

number

The number of spaces you want in the string.

Remarks: VBScript Space Function

The following example uses the **Space** function to return a string consisting of a specified number of spaces:

```
Dim MyString

MyString = Space(10)      ' Returns a string with 10 spaces.

MyString = "Hello" & Space(10) & "World" ' Insert 10 spaces between
two strings.
```

VBScript Split Function

Returns a zero-based, one-dimensional array containing a specified number of substrings.

Syntax: VBScript Split Function

```
Split(expression[, delimiter[, count[, compare]]])
```

expression

A string expression containing substrings and delimiters. If *expression* is a zero-length string, **Split** returns an empty array, that is, an array with no elements and no data. Required.

delimiter

A string character used to identify substring limits. If omitted, the space character (" ") is assumed to be the delimiter. If *delimiter* is a zero-length string, a single-element array containing the entire *expression* string is returned. Optional.

count

The number of substrings to be returned; -1 indicates that all substrings are returned. Optional.

compare

The numeric value indicating the kind of comparison to use when evaluating substrings. See Settings section for values. Optional.

Settings: VBScript Split Function

The *compare* argument can have the following values:

Constant	Value	Description
vbBinaryCompare	0	Perform a binary comparison.
vbTextCompare	1	Perform a textual comparison.

Remarks: VBScript Split Function

The following example uses the **Split** function to return an array from a string. The function performs a textual comparison of the delimiter, and returns all of the substrings.

```
Dim MyString, MyArray, Msg

MyString = "VBScriptXisXfun!"

MyArray = Split(MyString, "x", -1, 1)
```

```
' MyArray(0) contains "VBScript".
' MyArray(1) contains "is".
' MyArray(2) contains "fun!".
Msg = MyArray(0) & " " & MyArray(1)
Msg = Msg      & " " & MyArray(2)
MsgBox Msg
```

VBScript Sqr Function

Returns the square root of a number.

Syntax: VBScript Sqr Function

```
Sqr(number)
```

Arguments: VBScript Sqr Function

number

Any valid numeric expression greater than or equal to 0.

Remarks: VBScript Sqr Function

The following example uses the **Sqr** function to calculate the square root of a number:

```
Dim MySqr
MySqr = Sqr(4)       ' Returns 2.
MySqr = Sqr(23)       ' Returns 4.79583152331272.
MySqr = Sqr(0)       ' Returns 0.
MySqr = Sqr(-4)       ' Generates a run-time error.
```

VBScript StrComp Function

Returns a value indicating the result of a string comparison.

Syntax: VBScript StrComp Function

```
StrComp(string1, string2[, compare])
```

Arguments: VBScript StrComp Function

string1

Any valid string expression. Required.

string2

Any valid string expression. Required.

compare

The numeric value indicating the kind of comparison to use when evaluating strings. If omitted, a binary comparison is performed. See Settings section for values. Optional.

Settings: VBScript StrComp Function

The *compare* argument can have the following values:

Constant	Value	Description
vbBinaryCompare	0	Perform a binary comparison.
vbTextCompare	1	Perform a textual comparison.

Return Values: VBScript StrComp Function

The **StrComp** function has the following return values:

If	StrComp returns
string1 is less than *string2*	-1
string1 is equal to *string2*	0
string1 is greater than *string2*	1
string1 or *string2* is **Null**	Null

Remarks: VBScript StrComp Function

The following example uses the **StrComp** function to return the results of a string comparison. If the third argument is 1, a textual comparison is performed; if the third argument is 0 or omitted, a binary comparison is performed.

```
Dim MyStr1, MyStr2, MyComp
MyStr1 = "ABCD": MyStr2 = "abcd"      ' Define variables.
MyComp = StrComp(MyStr1, MyStr2, 1)      ' Returns 0.
MyComp = StrComp(MyStr1, MyStr2, 0)      ' Returns -1.
MyComp = StrComp(MyStr2, MyStr1)      ' Returns 1.
```

VBScript StrReverse Function

Returns a string in which the character order of a specified string is reversed.

Syntax: VBScript StrReverse Function

```
StrReverse(string1)
```

Arguments: VBScript StrReverse Function

string1

The string whose characters are to be reversed. If *string1* is a zero-length string (""), a zero-length string is returned. If *string1* is **Null**, an error occurs.

Remarks: VBScript StrReverse Function

The following example uses the **StrReverse** function to return a string in reverse order:

```
Dim MyStr
MyStr = StrReverse("VBScript") ' MyStr contains "tpircSBV".
```

VBScript String Function

Returns a repeating character string of the length specified.

Syntax: VBScript String Function

```
String(number, character)
```

Arguments: VBScript String Function

number

The length of the returned string. If *number* contains **Null**, **Null** is returned.

character

The character code specifying the character or string expression whose first character is used to build the return string. If *character* contains **Null**, **Null** is returned.

Remarks: VBScript String Function

If you specify a number for *character* greater than 255, **String** converts the number to a valid character code using the formula:

```
character Mod 256
```

VBScript Tan Function

Returns the tangent of an angle.

Syntax: VBScript Tan Function

```
Tan(number)
```

Arguments: VBScript Tan Function

number

Any valid numeric expression that expresses an angle in radians.

Remarks: VBScript Tan Function

Tan takes an angle and returns the ratio of two sides of a right triangle. The ratio is the length of the side opposite the angle divided by the length of the side adjacent to the angle.

To convert degrees to radians, multiply degrees by pi/180. To convert radians to degrees, multiply radians by 180/pi.

The following example uses the **Tan** function to return the tangent of an angle:

```
Dim MyAngle, MyCotangent

MyAngle = 1.3      ' Define angle in radians.

MyCotangent = 1 / Tan(MyAngle)      ' Calculate cotangent.
```

VBScript Time Function

Returns a **Variant** of subtype **Date** indicating the current system time.

Syntax: VBScript Time Function

```
Time()
```

Remarks: VBScript Time Function

The following example uses the **Time** function to return the current system time:

```
Dim MyTime

MyTime = Time      ' Return current system time.
```

VBScript TimeSerial Function

Returns a **Variant** of subtype **Date** containing the time for a specific hour, minute, and second.

Syntax: VBScript TimeSerial Function

```
TimeSerial(hour, minute, second)
```

Arguments: VBScript TimeSerial Function

hour

A number between 0 (12:00 A.M.) and 23 (11:00 P.M.), inclusive, or a numeric expression.

minute

Any numeric expression.

second

Any numeric expression.

Remarks: VBScript TimeSerial Function

To specify a time, such as 11:59:59, the range of numbers for each **TimeSerial** argument should be in the accepted range for the unit; that is, 0-23 for hours and 0-59 for minutes and seconds. However, you can also specify relative times for each argument using any numeric expression that represents some number of hours, minutes, or seconds before or after a certain time. The following example uses expressions instead of absolute time numbers. The **TimeSerial** function returns a time for 15 minutes before (-15) six hours before noon (12 - 6), or 5:45:00 A.M.

```
TimeSerial(12 - 6, -15, 0)
```

When any argument exceeds the accepted range for that argument, it increments to the next larger unit as appropriate. For example, if you specify 75 minutes, it is evaluated as one hour and 15 minutes. However, if any single argument is outside the range -32,768 to 32,767, or if the time specified by the three arguments, either directly or by expression, causes the date to fall outside the acceptable range of dates, an error occurs.

VBScript TimeValue Function

Returns a **Variant** of subtype **Date** containing the time.

Syntax: VBScript TimeValue Function

```
TimeValue(time)
```

Arguments: VBScript TimeValue Function

time

Usually a string expression representing a time from 0:00:00 (12:00:00 A.M.) to 23:59:59 (11:59:59 P.M.), inclusive. However, *time* can also be any expression that represents a time in that range. If *time* contains **Null**, **Null** is returned.

Remarks: VBScript TimeValue Function

You can enter valid times using a 12-hour or 24-hour clock. For example, "2:24PM" and "14:24" are both valid time arguments.

If the *time* argument contains date information, **TimeValue** doesn't return the date information. However, if *time* includes invalid date information, an error occurs.

The following example uses the **TimeValue** function to convert a string to a time. You can also use date literals to directly assign a time to a **Variant** (for example, MyTime = #4:35:17 PM#).

```
Dim MyTime

MyTime = TimeValue("4:35:17 PM")      ' MyTime contains 4:35:17 PM.
```

VBScript TypeName Function

Returns a string that provides **Variant** subtype information about a variable.

Syntax: VBScript TypeName Function

```
TypeName(varname)
```

Arguments: VBScript TypeName Function

varname

A required argument can be any variable.

Return Values: VBScript TypeName Function

The **TypeName** function has the following return values:

Value	Description
Byte	Byte value
Integer	Integer value
Long	Long integer value
Single	Single-precision floating-point value
Double	Double-precision floating-point value
Currency	Currency value
Decimal	Decimal value
Date	Date or time value
String	Character string value
Boolean	Boolean value; True or False
Empty	Uninitialized
Null	No valid data
<ltobject type>	Actual type name of an object
Object	Generic object
Unknown	Unknown object type
Nothing	Object variable that has been explicitly set to Nothing, or has been set to return the value of a function that returned Nothing
Error	Error
Variant()	A Variant array

Remarks: VBScript TypeName Function

The following example uses the **TypeName** function to return information about a variable:

```
Dim ArrayVar(4), MyType

NullVar = Null      ' Assign Null value.

MyType = TypeName("VBScript")      ' Returns "String".

MyType = TypeName(4)                        ' Returns "Integer".

MyType = TypeName(37.50)                ' Returns "Double".

MyType = TypeName(NullVar)             ' Returns "Null".

MyType = TypeName(ArrayVar)           ' Returns "Variant()".
```

VBScript UBound Function

Returns the largest available subscript for the indicated dimension of an array.

Syntax: VBScript UBound Function

```
UBound(arrayname[, dimension])
```

Arguments: VBScript UBound Function

arrayname

The name of the array variable; follows standard variable naming conventions. Required.

dimension

The whole number indicating which dimension's upper bound is returned. Use 1 for the first dimension, 2 for the second, and so on. If dimension is omitted, 1 is assumed. Optional.

Remarks: VBScript UBound Function

The **UBound** function is used with the **LBound** function to determine the size of an array. Use the **LBound** function to find the lower limit of an array dimension.

The default lower bound for any dimension is always 0. As a result, **UBound** returns the following values for an array with these dimensions:

```
Dim A(100,3,4)
```

Statement	Return Value
Ubound(A, 1)	99
Ubound(A, 2)	2
Ubound(A, 3)	3

VBScript UCase Function

Returns a string that has been converted to uppercase.

Syntax: VBScript UCase Function

```
UCase(string)
```

Arguments: VBScript UCase Function

string

Any valid string expression. If *string* contains **Null**, **Null** is returned.

Remarks: VBScript UCase Function

Only lowercase letters are converted to uppercase; all uppercase letters and nonletter characters remain unchanged.

The following example uses the **UCase** function to return an uppercase version of a string:

```
Dim MyWord

MyWord = UCase("Hello World")      ' Returns "HELLO WORLD".
```

VBScript VarType Function

Returns a value indicating the subtype of a variable.

Syntax: VBScript VarType Function

```
VarType(varname)
```

Arguments: VBScript VarType Function

varname

Any variable.

Return Values: VBScript VarType Function

The **VarType** function returns the following values:

Constant	Value	Description
vbEmpty	0	Empty (uninitialized)
vbNull	1	Null (no valid data)
vbInteger	2	Integer
vbLong	3	Long integer
vbSingle	4	Single-precision floating-point number
vbDouble	5	Double-precision floating-point number
vbCurrency	6	Currency
vbDate	7	Date
vbString	8	String
vbObject	9	Automation object
vbError	10	Error
vbBoolean	11	Boolean
vbVariant	12	Variant (used only with arrays of Variants)
vbDataObject	13	A data-access object
vbByte	17	Byte
vbArray	8192	Array

Note

These constants are specified by VBScript. As a result, the names can be used anywhere in your code in place of the actual values.

Remarks: VBScript VarType Function

The **VarType** function never returns the value for **Array** by itself. It is always added to some other value to indicate an array of a particular type. The value for **Variant** is only returned when it has been added to the value for **Array** to indicate that the argument to the **VarType** function is an array. For example, the value returned for an array of integers is calculated as 2 + 8192, or 8194. If an object has a default property, **VarType (***object***)** returns the type of its default property.

VBScript Weekday Function

Returns a whole number representing the day of the week.

Syntax: VBScript Weekday Function

```
Weekday(date, [firstdayofweek])
```

Arguments: VBScript Weekday Function

date

Any expression that can represent a date. If *date* contains **Null**, **Null** is returned.

firstdayofweek

A constant that specifies the first day of the week. If omitted, **vbSunday** is assumed.

Settings: VBScript Weekday Function

The *firstdayofweek* argument has these settings:

Constant	Value	Description
vbUseSystem	0	Use National Language Support (NLS) API setting.
vbSunday	1	Sunday
vbMonday	2	Monday
vbTuesday	3	Tuesday
vbWednesday	4	Wednesday
vbThursday	5	Thursday
vbFriday	6	Friday
vbSaturday	7	Saturday

Return Values: VBScript Weekday Function

The **Weekday** function can return any of these values:

Constant	Value	Description
vbSunday	1	Sunday
vbMonday	2	Monday

vbTuesday	3	Tuesday
vbWednesday	4	Wednesday
vbThursday	5	Thursday
vbFriday	6	Friday
vbSaturday	7	Saturday

Remarks: VBScript Weekday Function

The following example uses the **Weekday** function to obtain the day of the week from a specified date:

```
Dim MyDate, MyWeekDay
MyDate = #October 19, 1962#      ' Assign a date.
MyWeekDay = Weekday(MyDate)      ' MyWeekDay contains 6 because
MyDate represents a Friday.
```

VBScript WeekdayName Function

Returns a string indicating the specified day of the week.

Syntax: VBScript WeekdayName Function

```
WeekdayName (weekday, abbreviate, firstdayofweek)
```

Arguments: VBScript WeekdayName Function

weekday

The numeric designation for the day of the week. Numeric value of each day depends on setting of the firstdayofweek setting. Required.

abbreviate

A Boolean value that indicates if the weekday name is to be abbreviated. If omitted, the default is **False**, which means that the weekday name is not abbreviated. Optional.

firstdayofweek

The numeric value indicating the first day of the week. See Settings section for values. Optional.

Settings: VBScript WeekdayName Function

The firstdayofweek argument can have the following values:

Constant	Value	Description
vbUseSystem	0	Use National Language Support (NLS) API setting.
vbSunday	1	Sunday (default)
vbMonday	2	Monday

vbTuesday	3	Tuesday
vbWednesday	4	Wednesday
vbThursday	5	Thursday
vbFriday	6	Friday
vbSaturday	7	Saturday

Remarks: VBScript WeekdayName Function

The following example uses the **WeekDayName** function to return the specified day:

```
Dim MyDate
MyDate = WeekDayName(6, True)      ' MyDate contains Fri.
```

VBScript Year Function

Returns a whole number representing the year.

Syntax: VBScript Year Function

```
Year(date)
```

Arguments: VBScript Year Function

date

Any expression that can represent a date. If *date* contains **Null**, **Null** is returned.

Remarks: VBScript Year Function

The following example uses the **Year** function to obtain the year from a specified date:

```
Dim MyDate, MyYear
MyDate = #October 19, 1962#      ' Assign a date.
MyYear = Year(MyDate)                  ' MyYear contains 1962.
```

VBScript Objects and Collections

VBScript Dictionary Object	Stores data as key, item pairs
VBScript Drive Object	Provides access to the properties of a disk drive or network share.
VBScript Err Object	Contains information about run-time errors.
VBScript File Object	Provides access to the properties of a file.
VBScript FileSystemObject Object	Provides access to a computer's file system.

VBScript Folder Object	Provides access to the properties of a folder.
VBScript TextStream Object	Facilitates sequential access to a file.
VBScript Collections	Useful collections of other objects.

VBScript Dictionary Object

VBScript Dictionary Object

The **Dictionary** object stores data as key, item pairs.

Methods: VBScript Dictionary Object

VBScript Dictionary Object Add Method	Adds a key, item pair.
VBScript Dictionary Object Exists Method	Indicates if a specified key exists.
VBScript Dictionary Object Items Method	Returns an array containing all items in a **Dictionary** object.
VBScript Dictionary Object Keys Method	Returns an array containing all keys in a **Dictionary** object.
VBScript Dictionary Object Remove Method	Removes a key, item pair.
VBScript Dictionary Object RemoveAll Method	Removes all key, item pairs.

Properties: VBScript Dictionary Object

VBScript Dictionary Object CompareMode Property	The comparison mode for string keys.
VBScript Dictionary Object Count Property	The number of items in a **Dictionary** object.
VBScript Dictionary Object Item Property	An item for a key.
VBScript Dictionary Object Key Property	A key.

Syntax: VBScript Dictionary Object

Scripting.Dictionary

Remarks: VBScript Dictionary Object

A **Dictionary** object is the equivalent of a PERL associative array. Items, which can be any form of data, are stored in the array. Each item is associated with a unique key. The key is used to retrieve an individual item and is usually a integer or a string, but can be anything except an array.

The following code illustrates how to create a **Dictionary** object:

```
Dim d 'Create a variable

Set d = CreateObject("Scripting.Dictionary")
```

d.Add "a", "Athens" 'Add some keys and items

d.Add "b", "Belgrade"

d.Add "c", "Cairo"

...

VBScript Dictionary Object Add Method

Adds a key and item pair to a **Dictionary** object.

Syntax: VBScript Dictionary Object Add Method

```
object.Add key, item
```

Arguments: VBScript Dictionary Object Add Method

object

The name of a **Dictionary** object. Required.

key

The *key* associated with the item being added. Required.

item

The *item* associated with the *key* being added. Required.

Remarks: VBScript Dictionary Object Add Method

An error occurs if the *key* already exists.

VBScript Dictionary Object CompareMode Property

Sets and returns the comparison mode for comparing string keys in a **Dictionary** object.

Syntax: VBScript Dictionary Object CompareMode Property

```
object.CompareMode[ = compare]
```

Arguments: VBScript Dictionary Object CompareMode Property

object

The name of a **Dictionary** object. Required.

compare

A value representing the comparison mode used by functions such as **StrComp**. Optional.

Settings: VBScript Dictionary Object CompareMode Property

The *compare* argument has the following settings:

Constant	Value	Description
vbBinaryCompare	0	Perform a binary comparison.
vbTextCompare	1	Perform a textual comparison.

Remarks: VBScript Dictionary Object CompareMode Property

Values greater than 2 can be used to refer to comparisons using specific Locale IDs (LCID). An error occurs if you try to change the comparison mode of a **Dictionary** object that already contains data.

The **CompareMode** property uses the same values as the compare argument for the **StrComp** function.

The **CompareMode** property of a dictionary is only available in VBScript; it cannot be used in JScript.

VBScript Dictionary Object Count Property

Returns the number of items in a **Dictionary** object. Read-only.

Syntax: VBScript Dictionary Object Count Property

```
object.Count
```

Arguments: VBScript Dictionary Object Count Property

object

The name of a **Dictionary** object.

Remarks: VBScript Dictionary Object Count Property

The following code illustrates use of the **Count** property:

```
Dim a, d, i 'Create some variables
Set d = CreateObject("Scripting.Dictionary")
d.Add "a", "Athens" 'Add some keys and items.
d.Add "b", "Belgrade"
d.Add "c", "Cairo"
a = d.Keys 'Get the keys
For i = 0 To d.Count -1 'Iterate the array
Print a(i) 'Print key
Next
...
```

VBScript Dictionary Object Exists Method

Returns **True** if a specified key exists in the **Dictionary** object, **False** if it does not.

Syntax: VBScript Dictionary Object Exists Method

```
object.Exists(key)
```

Arguments: VBScript Dictionary Object Exists Method

object

The name of a **Dictionary** object. Required.

key

The key value being searched for in the **Dictionary** object. Required.

VBScript Dictionary Object Item Property

Sets or returns an item for a specified key in a **Dictionary** object.

Syntax: VBScript Dictionary Object Item Property

```
object.Item(key) [= newitem]
```

Arguments: VBScript Dictionary Object Item Property

object

The name of a **Dictionary** object. Required.

key

A key associated with the item being retrieved or added. Required.

newitem

If provided, *newitem* is the new value associated with the specified key. Optional.

Remarks: VBScript Dictionary Object Item Property

If *key* is not found when changing an *item,* a new *key* is created with the specified *newitem.* If *key* is not found when attempting to return an existing *item,* a new *key* is created and its corresponding *item* is left empty.

VBScript Dictionary Object Items Method

Returns an array containing all the items in a **Dictionary** object.

Syntax: VBScript Dictionary Object Items Method

```
object.Items
```

Arguments: VBScript Dictionary Object Items Method

object

The name of a **Dictionary** object.

Remarks: VBScript Dictionary Object Items Method

The following code illustrates use of the **Items** method:

```
Dim a, d, i 'Create some variables
Set d = CreateObject("Scripting.Dictionary")
d.Add "a", "Athens" 'Add some keys and items
d.Add "b", "Belgrade"
d.Add "c", "Cairo"
a = d.Items 'Get the items
For i = 0 To d.Count -1 'Iterate the array
Response.Write a(i) 'Print item
Next
...
```

VBScript Dictionary Object Key Property

Sets a key in a **Dictionary** object.

Syntax: VBScript Dictionary Object Key Property

```
object.Key(key) = newkey
```

Arguments: VBScript Dictionary Object Key Property

object

The name of a **Dictionary** object. Required.

key

Key value being changed. Required.

newkey

New value that replaces the specified *key.* Required.

Remarks: VBScript Dictionary Object Key Property

If *key* is not found when changing a *key,* an error will occur.

VBScript Dictionary Object Keys Method

Returns an array containing all existing keys in a **Dictionary** object.

Syntax: VBScript Dictionary Object Keys Method

```
object.Keys
```

Arguments: VBScript Dictionary Object Keys Method

object

The name of a **Dictionary** object.

Remarks: VBScript Dictionary Object Keys Method

The following code illustrates use of the **Keys** method:

```
Dim a, d, i 'Create some variables
Set d = CreateObject("Scripting.Dictionary")
d.Add "a", "Athens" 'Add some keys and items.
d.Add "b", "Belgrade"
d.Add "c", "Cairo"
a = d.keys 'Get the keys
For i = 0 To d.Count -1 'Iterate the array
Response.write a(i) 'Print key
Next
...
```

VBScript Dictionary Object Remove Method

Removes a key, item pair from a **Dictionary** object.

Syntax: VBScript Dictionary Object Remove Method

```
object.Remove(key)
```

Arguments: VBScript Dictionary Object Remove Method

object

The name of a **Dictionary** object. Required.

key

The *key* associated with the key, item pair you want to remove from the **Dictionary** object. Required.

Remarks: VBScript Dictionary Object Remove Method

An error occurs if the specified key, item pair does not exist.

The following code illustrates use of the **Remove** method:

```
Dim a, d, i 'Create some variables
Set d = CreateObject("Scripting.Dictionary")
d.Add "a", "Athens" 'Add some keys and items
d.Add "b", "Belgrade"
```

```
d.Add "c", "Cairo"
...
a = d.Remove("b") 'Remove second pair
```

VBScript Dictionary Object RemoveAll Method

The **RemoveAll** method removes all key, item pairs from a **Dictionary** object.

Syntax: VBScript Dictionary Object RemoveAll Method

```
object.RemoveAll
```

Arguments: VBScript Dictionary Object RemoveAll Method

object

The name of a **Dictionary** object.

Remarks: VBScript Dictionary Object RemoveAll Method

The following code illustrates use of the **RemoveAll** method:

```
Dim a, d, i 'Create some variables
Set d = CreateObject("Scripting.Dictionary")
d.Add "a", "Athens" 'Add some keys and items
d.Add "b", "Belgrade"
d.Add "c", "Cairo"
...
a = d.RemoveAll 'Clear the dictionary
```

VBScript Drive Object

VBScript Drive Object

The VBScript **Drive** object provides access to the properties of a particular disk drive or network share.

Properties: VBScript Drive Object

VBScript Drive Object AvailableSpace Property	The amount of space available to a user on the specified drive or network share. *This property is not currently supported on UNIX.*
VBScript Drive Object DriveLetter Property	The drive letter of a physical local drive or network share. *This property is not currently supported on UNIX.*
VBScript Drive Object DriveType	A value indicating the type of a drive. *This property is*

Property	*not currently supported on UNIX.*
VBScript Drive Object FileSystem Property	The type of file system in use for the drive. *This property is non currently supported on UNIX.*
VBScript Drive Object FreeSpace Property	The amount of free space available to a user on the drive or network share. *This property is not currently supported on UNIX.*
VBScript Drive Object IsReady Property	**True** if the drive is ready, **False** if not.
VBScript Drive Object Path Property	The file system path for a drive.
VBScript Drive Object RootFolder Property	A **Folder** object representing the root folder of a drive.
VBScript Drive Object SerialNumber Property	The decimal serial number used to uniquely identify the disk volume. *This property is not currently supported on UNIX.*
VBScript Drive Object ShareName Property	The network share name of a drive. *This property is not currently supported on UNIX.*
VBScript Drive Object TotalSize Property	The total space, in bytes, of a drive or network share. *This property is not currently supported on UNIX.*
VBScript Drive Object VolumeName Property	The volume name of a drive. *This property is not currently supported on UNIX.*

Remarks: VBScript Drive Object

The following code illustrates the use of the **Drive** object to access drive properties:

```
Sub ShowFreeSpace(drvPath)
Dim fs, d, s
Set fs = CreateObject("Scripting.FileSystemObject")
Set d = fs.GetDrive(fs.GetDriveName(drvPath))
s = "Drive " & UCase(drvPath) & " - "
s = s & d.VolumeName & vbCrLf
s = s & "Free Space: " & FormatNumber(d.FreeSpace/1024, 0)
s = s & " Kbytes"
Response.Write s
End Sub
```

VBScript Drive Object AvailableSpace Property

Returns the amount of space available to a user on the specified drive or network share. *This property is not currently available on UNIX systems.*

Syntax: VBScript Drive Object AvailableSpace Property

```
object.AvailableSpace
```

Arguments: VBScript Drive Object AvailableSpace Property

object

A **Drive** object.

Remarks: VBScript Drive Object AvailableSpace Property

The value returned by the **AvailableSpace** property is typically the same as that returned by the Drive Object FreeSpace Property. Differences may occur between the two for computer systems that support quotas. The following code illustrates the use of the **AvailableSpace** property:

```
Sub ShowAvailableSpace(drvPath)
Dim fs, d, s
Set fs = CreateObject("Scripting.FileSystemObject")
Set d = fs.GetDrive(fs.GetDriveName(drvPath))
s = "Drive " & UCase(drvPath) & " - "
s = s & d.VolumeName & vbCrLf
s = s & "Available Space: " & FormatNumber(d.AvailableSpace/1024, 0)
s = s & " Kbytes"
Response.Write s
End Sub
```

VBScript Drive Object DriveLetter Property

Returns the drive letter of a physical local drive or a network share. *This property is not currently available on Unix systems.* Read-only.

Syntax: VBScript Drive Object DriveLetter Property

```
object.DriveLetter
```

Arguments: VBScript Drive Object DriveLetter Property

object

A **Drive** object.

Remarks: VBScript Drive Object DriveLetter Property

The **DriveLetter** property returns a zero-length string ("") if the specified drive is not associated with a drive letter, for example, a network share that has not been mapped to a drive letter.

The following code illustrates the use of the **DriveLetter** property:

```
Sub ShowDriveLetter(drvPath)
```

```
Dim fs, d, s
Set fs = CreateObject("Scripting.FileSystemObject")
Set d = fs.GetDrive(fs.GetDriveName(drvPath))
s = "Drive " & d.DriveLetter & ": - "
s = s & d.VolumeName & vbCrLf
s = s & "Free Space: " & FormatNumber(d.FreeSpace/1024, 0)
s = s & " Kbytes"
Response.Write s
End Sub
```

VBScript Drive Object DriveType Property

Returns a value indicating the type of a specified drive. *This property is not currently supported on UNIX.*

Syntax: VBScript Drive Object DriveType Property

```
object.DriveType
```

Arguments: VBScript Drive Object DriveType Property

object

A **Drive** object.

Remarks: VBScript Drive Object DriveType Property

The following code illustrates the use of the **DriveType** property:

```
Sub ShowDriveType(drvpath)
Dim fs, d, s, t
Set fs = CreateObject("Scripting.FileSystemObject")
Set d = fs.GetDrive(drvpath)
Select Case d.DriveType
Case 0: t = "Unknown"
Case 1: t = "Removable"
Case 2: t = "Fixed"
Case 3: t = "Network"
Case 4: t = "CD-ROM"
Case 5: t = "RAM Disk"
End Select
```

```
s = "Drive " & d.DriveLetter & ": - " & t
Response.Write s
End Sub
```

VBScript Drive Object FileSystem Property

Returns the type of file system in use for the specified drive. *This property is not currently supported on UNIX.*

Syntax: VBScript Drive Object FileSystem Property

```
object.FileSystem
```

Arguments: VBScript Drive Object FileSystem Property

object

A **Drive** object.

Remarks: VBScript Drive Object FileSystem Property

Available return types include FAT, NTFS, and CDFS.

The following code illustrates the use of the **FileSystem** property:

```
Sub ShowFileSystemType
Dim fs,d, s
Set fs = CreateObject("Scripting.FileSystemObject")
Set d = fs.GetDrive("e:")
s = d.FileSystem
Response.Write s
End Sub
```

VBScript Drive Object FreeSpace Property

Returns the amount of free space available to a user on the specified drive or network share. *This property is not currently supported on UNIX.* Read-only.

Syntax: VBScript Drive Object FreeSpace Property

```
object.FreeSpace
```

Arguments: VBScript Drive Object FreeSpace Property

object

A **Drive** object.

Remarks: VBScript Drive Object FreeSpace Property

The value returned by the **FreeSpace** property is typically the same as that returned by the Drive Object AvailableSpace Property. Differences may occur between the two for computer systems that support quotas.

The following code illustrates the use of the **FreeSpace** property:

```
Sub ShowFreeSpace(drvPath)
Dim fs, d, s
Set fs = CreateObject("Scripting.FileSystemObject")
Set d = fs.GetDrive(fs.GetDriveName(drvPath))
s = "Drive " & UCase(drvPath) & " - "
s = s & d.VolumeName & vbCrLf
s = s & "Free Space: " & FormatNumber(d.FreeSpace/1024, 0)
s = s & " Kbytes"
Response.Write s
End Sub
```

VBScript Drive Object IsReady Property

Returns **True** if the specified drive is ready; **False** if it is not.

Syntax: VBScript Drive Object IsReady Property

```
object.IsReady
```

Arguments: VBScript Drive Object IsReady Property

object

A **Drive** object.

Remarks: VBScript Drive Object IsReady Property

For removable-media drives and CD-ROM drives, **IsReady** returns **True** only when the appropriate media is inserted and ready for access. On UNIX systems the **IsReady** property always returns **True**.

The following code illustrates the use of the **IsReady** property:

```
Sub ShowDriveInfo(drvpath)
Dim fs, d, s, t
Set fs = CreateObject("Scripting.FileSystemObject")
Set d = fs.GetDrive(drvpath)
Select Case d.DriveType
Case 0: t = "Unknown"
```

```
Case 1: t = "Removable"
Case 2: t = "Fixed"
Case 3: t = "Network"
Case 4: t = "CD-ROM"
Case 5: t = "RAM Disk"
End Select
s = "Drive " & d.DriveLetter & ": - " & t
If d.IsReady Then
s = s & vbCrLf & "Drive is Ready."
Else
s = s & vbCrLf & "Drive is not Ready."
End If
Response.Write s
End Sub
```

Under UNIX, the **IsReady** property is always **True**.

VBScript Drive Object Path Property

Returns the path for a specified drive.

Syntax: VBScript Drive Object Path Property

```
object.Path
```

Arguments: VBScript Drive Object Path Property

object

A **Drive** object.

Remarks: VBScript Drive Object Path Property

For drive letters, the root drive is not included. For example, the path for the C drive is C:, not C:\.

VBScript Drive Object RootFolder Property

Returns a **Folder** object representing the root folder of a specified drive. Read-only.

Syntax: VBScript Drive Object RootFolder Property

```
object.RootFolder
```

Arguments: VBScript Drive Object RootFolder Property

object

A **Drive** object.

Remarks:[0] VBScript Drive Object RootFolder Property

All the files and folders contained on the drive can be accessed using the returned **Folder** object.

VBScript Drive Object SerialNumber Property

Returns the decimal serial number used to uniquely identify a disk volume. *This property is not currently supported on UNIX.*

Syntax: VBScript Drive Object SerialNumber Property

```
object.SerialNumber
```

Arguments: VBScript Drive Object SerialNumber Property

object

A **Drive** object.

Remarks: VBScript Drive Object SerialNumber Property

You can use the **SerialNumber** property to ensure that the correct disk is inserted in a drive with removable media.

The following code illustrates the use of the **SerialNumber** property:

```
Sub ShowDriveInfo(drvpath)

Dim fs, d, s, t

Set fs = CreateObject("Scripting.FileSystemObject")

Set d =
fs.GetDrive(fs.GetDriveName(fs.GetAbsolutepathname(drvpath)))

Select Case d.DriveType

Case 0: t = "Unknown"

Case 1: t = "Removable"

Case 2: t = "Fixed"

Case 3: t = "Network"

Case 4: t = "CD-ROM"

Case 5: t = "RAM Disk"

End Select

s = "Drive " & d.DriveLetter & ": - " & t

s = s & vbCrLf & "SN: " & d.SerialNumber

Response.Write s
```

```
End Sub
```

VBScript Drive Object ShareName Property

Returns the network share name for a specified drive. *This property is not currently supported on UNIX.*

Syntax: VBScript Drive Object ShareName Property

```
object.ShareName
```

Arguments: VBScript Drive Object ShareName Property

object

A **Drive** object.

Remarks: VBScript Drive Object ShareName Property

If object is not a network drive, the **ShareName** property returns a zero-length string (""). The following code illustrates the use of the **ShareName** property:

```
Sub ShowDriveInfo(drvpath)
Dim fs, d, s
Set fs = CreateObject("Scripting.FileSystemObject")
Set d =
fs.GetDrive(fs.GetDriveName(fs.GetAbsolutepathname(drvpath)))
s = "Drive " & d.DriveLetter & ": - " & d.ShareName
Response.Write s
End Sub
```

VBScript Drive Object TotalSize Property

Returns the total space, in bytes, of a drive or network share. *This property is not currently supported on UNIX.*

Syntax: VBScript Drive Object TotalSize Property

```
object.TotalSize
```

Arguments: VBScript Drive Object TotalSize Property

object

A **Drive** object.

Remarks: VBScript Drive Object TotalSize Property

The following code illustrates the use of the **TotalSize** property:

```
Sub ShowSpaceInfo(drvpath)
```

```
Dim fs, d, s

Set fs = CreateObject("Scripting.FileSystemObject")

Set d =
fs.GetDrive(fs.GetDriveName(fs.GetAbsolutepathname(drvpath)))

s = "Drive " & d.DriveLetter & ":"

s = s & vbCrLf

s = s & "Total Size: " & FormatNumber(d.TotalSize/1024, 0) & "
Kbytes"

s = s & vbCrLf

s = s & "Available: " & FormatNumber(d.AvailableSpace/1024, 0) & "
Kbytes"

Response.Write s

End Sub
```

VBScript Drive Object VolumeName Property

Sets or returns the volume name of the specified drive. *This property is not currently supported on UNIX.* Read/write.

Syntax: VBScript Drive Object VolumeName Property

```
object.VolumeName [= newname]
```

Arguments: VBScript Drive Object VolumeName Property

object

The name of a **Drive** object. Required.

newname

If provided, newname is the new name of the specified object. Optional.

Remarks: VBScript Drive Object VolumeName Property

The following code illustrates the use of the **VolumeName** property:

```
Sub ShowVolumeInfo(drvpath)

Dim fs, d, s

Set fs = CreateObject("Scripting.FileSystemObject")

Set d =
fs.GetDrive(fs.GetDriveName(fs.GetAbsolutepathname(drvpath)))

s = "Drive " & d.DriveLetter & ": - " & d.VolumeName

Response.Write s
```

```
End Sub
```

VBScript Err Object

VBScript Err Object

The VBScript **Err** object contains information about run-time errors.

Methods: VBScript Err Object

VBScript Err Object Clear Method	Clears all property settings.
VBScript Err Object Raise Method	Generate a run-time error.

Properties: VBScript Err Object

VBScript Err Object Description Property	The descriptive string associated with an error.
VBScript Err Object HelpContext Property	A context ID for a topic in a Windows help file.
VBScript Err Object HelpFile Property	A fully qualified path to a Windows help file.
VBScript Err Object Number Property	A numeric value identifying an error.
VBScript Err Object Source Property	The name of the object or application that originally generated the error.

Remarks: VBScript Err Object

The properties of the **Err** object are set by the generator of an error—Visual Basic, an Automation object, or the VBScript programmer.

The default property of the **Err** object is **Number**. **Err.Number** contains an integer and can be used by an Automation object to return an SCODE.

When a run-time error occurs, the properties of the **Err** object are filled with information that uniquely identifies the error and information that can be used to handle it. To generate a run-time error in your code, use the VBScript Err Object Raise Method. The **Err** object's properties are reset to zero or zero-length strings ("") after an **On Error Resume Next** statement. The VBScript Err Object Clear Method can be used to explicitly reset **Err**.

The **Err** object is an intrinsic object with global scope—there is no need to create an instance of it in your code.

VBScript Err Object Clear Method

Clears all property settings of the **Err** object.

Syntax: VBScript Err Object Clear Method

```
Err.Clear
```

Remarks: VBScript Err Object Clear Method

Use **Clear** to explicitly clear the **Err** object after an error has been handled. This is necessary, for example, when you use deferred error handling with **On Error Resume Next**. VBScript calls the **Clear** method automatically whenever any of the following statements is executed:

- On Error Resume Next
- Exit Sub
- Exit Function

VBScript Err Object Description Property

Returns or sets a descriptive string associated with an error.

Syntax: VBScript Err Object Description Property

```
Err.Description [= stringexpression]
```

Arguments: VBScript Err Object Description Property

stringexpression

A string expression containing a description of the error.

Remarks: VBScript Err Object Description Property

The **Description** property consists of a short description of the error. Use this property to alert the user to an error that you can't or don't want to handle. When generating a user-defined error, assign a short description of your error to this property. If **Description** isn't filled in, and the value of theVBScript Err Object Number Property corresponds to a VBScript run-time error, the descriptive string associated with the error is returned.

VBScript Err Object HelpContext Property

Sets or returns a context ID for a topic in a Help File.

Syntax: VBScript Err Object HelpContext Property

```
Err.HelpContext [= contextID]
```

Arguments: VBScript Err Object HelpContext Property

contextID

A valid identifier for a Help topic within the Help file. Optional.

Remarks: VBScript Err Object HelpContext Property

If a Help file is specified in VBScript Err Object HelpFile Property, the **HelpContext** property is used to automatically display the Help topic identified. If both **HelpFile** and **HelpContext** are

empty, the value of the VBScript Err Object Number Property is checked. If it corresponds to a VBScript run-time error value, then the VBScript Help context ID for the error is used. If the **Number** property doesn't correspond to a VBScript error, the contents screen for the VBScript Help file is displayed.

VBScript Err Object HelpFile Property

Sets or returns a fully qualified path to a Help File.

Syntax: VBScript Err Object HelpFile Property

```
Err.HelpFile [= contextID]
```

Arguments: VBScript Err Object HelpFile Property

contextID

The fully qualified path to the Help file. Optional.

Remarks: VBScript Err Object HelpFile Property

If a Help file is specified in **HelpFile**, it is automatically called when the user clicks the Help button (or presses the F1 key) in the error message dialog box. If the VBScript Err Object HelpContext Property contains a valid context ID for the specified file, that topic is automatically displayed. If no **HelpFile** is specified, the VBScript Help file is displayed.

VBScript Err Object Number Property

Returns or sets a numeric value specifying an error. **Number** is the **Err** object's default property.

Syntax: VBScript Err Object Number Property

```
Err.Number [= errornumber]
```

Arguments: VBScript Err Object Number Property

errornumber

An integer representing a VBScript error number or an SCODE error value.

Remarks: VBScript Err Object Number Property

When returning a user-defined error from an Automation object, set **Err.Number** by adding the number you selected as an error code to the constant **vbObjectError**. For example, you use the following code to return the number 1051 as an error code:

```
Err.Raise Number:= vbObjectError + 1051, Source:= "SomeClass"
```

VBScript Err Object Raise Method

Generates a run-time error.

Syntax: VBScript Err Object Raise Method

```
Err.Raise(number, source, description, helpfile, helpcontext)
```

Arguments: VBScript Err Object Raise Method

number

A **Long** integer subtype that identifies the nature of the error. VBScript errors (both VBScript-defined and user-defined errors) are in the range 0-65535.

source

A string expression naming the object or application that originally generated the error. When setting this property for an Automation object, use the form *project.class.* If nothing is specified, the programmatic ID of the current VBScript project is used.

description

A string expression describing the error. If unspecified, the value in *number* is examined. If it can be mapped to a VBScript run-time error code, a string provided by VBScript is used as *description.* If there is no VBScript error corresponding to *number,* a generic error message is used.

helpfile

The fully qualified path to the Help file in which help on this error can be found. If unspecified, VBScript uses the fully qualified drive, path, and file name of the VBScript Help file.

helpcontext

The context ID identifying a topic within *helpfile* that provides help for the error. If omitted, the VBScript Help file context ID for the error corresponding to the number property is used, if it exists.

Remarks: VBScript Err Object Raise Method

All the argument are optional except *number.* If you use **Raise**, however, without specifying some arguments, and the property settings of the **Err** object contain values that have not been cleared, those values become the values for your error.

When setting the *number* property to your own error code in an Automation object, you add your error code number to the constant **vbObjectError.** For example, to generate the error number 1050, assign **vbObjectError** + 1050 to the *number* property.

VBScript Err Object Source Property

Returns or sets the name of the object or application that originally generated the error.

Syntax: VBScript Err Object Source Property

```
Err.Source [= stringexpression]
```

Arguments: VBScript Err Object Source Property

stringexpression

A string expression representing the application that generated the error.

Remarks: VBScript Err Object Source Property

The **Source** property specifies a string expression that is usually the class name or programmatic ID of the object that caused the error. Use **Source** to provide your users with information when your code is unable to handle an error generated in an accessed object. For example, if you access Microsoft Excel and it generates a *Division by zero* error, Microsoft Excel sets Err Object Number Property to its error code for that error and sets **Source** to Excel.Application. Note that if the error is generated in another object called by Microsoft Excel, Excel intercepts the error and sets **Err.Number** to its own code for *Division by zero*. However, it leaves the other **Err** object (including **Source**) as set by the object that generated the error.

Source always contains the name of the object that originally generated the error—your code can try to handle the error according to the error documentation of the object you accessed. If your error handler fails, you can use the **Err** object information to describe the error to your user, using **Source** and the other **Err** to inform the user which object originally caused the error, its description of the error, and so forth.

VBScript File Object

VBScript File Object

The VBScript **File** object provides access to all the properties of a file.

Methods: VBScript File Object

VBScript File Object Copy Method	Copies a file from one location to another.
VBScript File Object Delete Method	Deletes a file.
VBScript File Object Move Method	Moves a file from one location to another.
VBScript File Object OpenAsTextStream Method	Opens a file and returns a **TextStream** object.

Properties: VBScript File Object

VBScript File Object Attributes Property	The attributes of a file.
VBScript File Object DateCreated Property	The date and time that the file was created.
VBScript File Object DateLastAccessed Property	The date and time that the file was last accessed.
VBScript File Object DateLastModified Property	The date and time that the file was last modified.
VBScript File Object Drive Property	The drive letter of the drive on which the file resides.
VBScript File Object Name Property	The name of the file.
VBScript File Object ParentFolder	The **Folder** object for the parent of the file.

Property	
VBScript File Object Path Property	The file system path to the file.
VBScript File Object ShortName Property	The short name used by programs that require 8.3 names. *This property is not currently supported on UNIX.*
VBScript File Object ShortPath Property	The short path use by programs that require 8.3 names. *This property is not currently supported on UNIX.*
VBScript File Object Size Property	The size, in bytes, of a file.
VBScript File Object Type Property	Information about the type of a file. *This property is not currently supported on UNIX.*

Remarks: VBScript File Object

The following code illustrates how to obtain a **File** object and how to view one of its properties.

```
Sub ShowFileInfo(filespec)
Dim fs, f, s
Set fs = CreateObject("Scripting.FileSystemObject")
Set f = fs.GetFile(filespec)
s = f.DateCreated
Response.Write s
End Sub
```

VBScript File Object Attributes Property

Sets or returns the attributes of files. Read/write or read-only, depending on the attribute.

Note

This property depends on the underlying operating system for its behavior. If the OS file system does not support the file attribute requested, an error will be returned.

Syntax: VBScript File Object Attributes Property

```
object.Attributes [= newattributes]
```

Arguments: VBScript File Object Attributes Property

object

The name of a **File** object. Required.

newattributes

The new value for the attributes of the specified object. Optional.

Settings: VBScript File Object Attributes Property

The newattributes argument can have any of the following values or any logical combination of the following values:

Constant	Value	Description
Normal	0	Normal file. No attributes are set.
ReadOnly	1	Read-only file. Attribute is read/write.
Hidden	2	Hidden file. Attribute is read/write.
System	4	System file. Attribute is read/write.
Volume	8	Disk drive volume label. Attribute is read-only.
Directory	16	Folder or directory. Attribute is read-only.
Archive	32	File has changed since last backup. Attribute is read/write.
Alias	64	Link or shortcut. Attribute is read-only.
Compressed	128	Compressed file. Attribute is read-only.

Remarks: VBScript File Object Attributes Property

The following code illustrates the use of the **Attributes** property with a file:

```
Sub SetClearArchiveBit(filespec)

Dim fs, f, r

Set fs = CreateObject("Scripting.FileSystemObject")

Set f = fs.GetFile(fs.GetFileName(filespec))

If f.attributes and 32 Then

r = MsgBox("The Archive bit is set, do you want to clear it?",
vbYesNo, "Set/Clear Archive Bit")

If r = vbYes Then

f.attributes = f.attributes - 32

MsgBox "Archive bit is cleared."

Else

MsgBox "Archive bit remains set."

End If

Else

r = MsgBox("The Archive bit is not set. Do you want to set it?",
vbYesNo, "Set/Clear Archive Bit")

If r = vbYes Then
```

```
f.attributes = f.attributes + 32
MsgBox "Archive bit is set."
Else
MsgBox "Archive bit remains clear."
End If
End If
End Sub
```

VBScript File Object Copy Method

Copies a specified file from one location to another.

Syntax: VBScript File Object Copy Method

```
object.Copy destination[, overwrite]
```

Arguments: VBScript File Object Copy Method

object

The name of a **File** object. Required.

destination

The destination where the file is to be copied. Wildcard characters are not allowed. Required.

overwrite

A Boolean value that is **True** (default) if existing files are to be overwritten; **False** if they are not. Optional.

Remarks: VBScript File Object Copy Method

The results of the **Copy** method on a **File** object are identical to operations performed using **CopyFile** where the file referred to by *object* is passed as an argument. You should note, however, that the alternative method is capable of copying multiple files.

VBScript File Object DateCreated Property

Returns the date and time that the specified file was created. Read-only.

Syntax: VBScript File Object DateCreated Property

```
object.DateCreated
```

Arguments: VBScript File Object DateCreated Property

object

A **File** object.

Remarks: VBScript File Object DateCreated Property

The following code illustrates the use of the **DateCreated** property with a file:

```
Sub ShowFileInfo(filespec)
Dim fs, f, s
Set fs = CreateObject("Scripting.FileSystemObject")
Set f = fs.GetFile(filespec)
s = "Created: " & f.DateCreated
Response.Write s
End Sub
```

VBScript File Object DateLastAccessed Property

Returns the date and time that the specified file was last accessed. Read-only.

Syntax: VBScript File Object DateLastAccessed Property

object.**DateLastAccessed**

Arguments: VBScript File Object DateLastAccessed Property

object

A **File** object.

Remarks: VBScript File Object DateLastAccessed Property

The following code illustrates the use of the **DateLastAccessed** property with a file:

```
Sub ShowFileAccessInfo(filespec)
Dim fs, f, s
Set fs = CreateObject("Scripting.FileSystemObject")
Set f = fs.GetFile(filespec)
s = UCase(filespec) & vbCrLf
s = s & "Created: " & f.DateCreated & vbCrLf
s = s & "Last Accessed: " & f.DateLastAccessed & vbCrLf
s = s & "Last Modified: " & f.DateLastModified
Response.Write s, 0, "File Access Info"
End Sub
```

Note

This method depends on the underlying operating system for its behavior. If the operating system does not support providing time information, none will be returned.

VBScript File Object DateLastModified Property

Returns the date and time that the file was last modified. Read-only.

Syntax: VBScript File Object DateLastModified Property

```
object.DateLastModified
```

Arguments: VBScript File Object DateLastModified Property

object

A **File** object.

Remarks: VBScript File Object DateLastModified Property

The following code illustrates the use of the **DateLastModified** property with a file:

```
Sub ShowFileAccessInfo(filespec)
Dim fs, f, s
Set fs = CreateObject("Scripting.FileSystemObject")
Set f = fs.GetFile(filespec)
s = UCase(filespec) & vbCrLf
s = s & "Created: " & f.DateCreated & vbCrLf
s = s & "Last Accessed: " & f.DateLastAccessed & vbCrLf
s = s & "Last Modified: " & f.DateLastModified
Response.Write s, 0, "File Access Info"
End Sub
```

VBScript File Object Delete Method

Deletes a specified file.

Syntax: VBScript File Object Delete Method

```
object.Delete force
```

Arguments: VBScript File Object Delete Method

object

The name of a **File** object. Required.

force

A Boolean value that is **True** if files with the read-only attribute set are to be deleted; **False** (default) if they are not. Optional.

Remarks: VBScript File Object Delete Method

An error occurs if the specified file does not exist.

The results of the **Delete** method on a **File** object are identical to operations performed using **DeleteFile.**

VBScript File Object Drive Property

Returns the drive letter of the drive on which the specified file resides. Read-only.

Syntax: VBScript File Object Drive Property

```
object.Drive
```

Arguments: VBScript File Object Drive Property

object

A **File** object.

Remarks: VBScript File Object Drive Property

The following code illustrates the use of the **Drive** property:

```
Sub ShowFileAccessInfo(filespec)
Dim fs, f, s
Set fs = CreateObject("Scripting.FileSystemObject")
Set f = fs.GetFile(filespec)
s = f.Name & " on Drive " & UCase(f.Drive) & vbCrLf
s = s & "Created: " & f.DateCreated & vbCrLf
s = s & "Last Accessed: " & f.DateLastAccessed & vbCrLf
s = s & "Last Modified: " & f.DateLastModified
Response.Write s, 0, "File Access Info"
End Sub
```

Under UNIX, the **Drive** property is always "/".

VBScript File Object Move Method

Moves a specified file from one location to another.

Syntax: VBScript File Object Move Method

```
object.Move destination
```

Arguments: VBScript File Object Move Method

object

Required. Always the name of a **File** or **Folder** object.

destination

Required. Destination where the file is to be moved. Wildcard characters are not allowed.

Remarks: VBScript File Object Move Method

The results of the **Move** method on a **File** are identical to operations performed using **MoveFile**. You should note, however, that the alternative method is capable of moving multiple files or folders.

VBScript File Object Name Property

Sets or returns the name of a specified file. Read/write.

Syntax: VBScript File Object Name Property

```
object.Name [= newname]
```

Arguments: VBScript File Object Name Property

object

The name of a **File** object. Required.

newname

If provided, *newname* is the new name of the specified object. Optional.

Remarks: VBScript File Object Name Property

The following code illustrates the use of the **Name** property:

```
Sub ShowFileAccessInfo(filespec)
Dim fs, f, s
Set fs = CreateObject("Scripting.FileSystemObject")
Set f = fs.GetFile(filespec)
s = f.Name & " on Drive " & UCase(f.Drive) & vbCrLf
s = s & "Created: " & f.DateCreated & vbCrLf
s = s & "Last Accessed: " & f.DateLastAccessed & vbCrLf
s = s & "Last Modified: " & f.DateLastModified
Response.Write s, 0, "File Access Info"
End Sub
```

VBScript File Object OpenAsTextStream Method

Opens a specified file and returns a **TextStream** object that can be used to read from, write to, or append to the file.

Syntax: VBScript File Object OpenAsTextStream Method

```
object.OpenAsTextStream([iomode, [format]])
```

Arguments: VBScript File Object OpenAsTextStream Method

object

The name of a **File** object. Required.

iomode

Indicates input/output mode. Can be one of three constants: **ForReading, ForWriting,** or **ForAppending**. Optional.

format

One of three **Tristate** values used to indicate the format of the opened file. If omitted, the file is opened as ASCII. Optional.

Settings: VBScript File Object OpenAsTextStream Method

The *iomode* arguments can have any of the following settings:

Constant	Value	Description
ForReading	1	Open a file for reading only. You can't write to this file.
ForWriting	2	Open a file for writing. If a file with the same name exists, its previous contents are overwritten.
ForAppending	8	Open a file and write to the end of the file.

The format *argument* can have any of the following settings:

Constant	Value	Description
TristateUseDefault	-2	Opens the file using the system default.
TristateTrue	-1	Opens the file as Unicode.
TristateFalse	0	Opens the file as ASCII.

Remarks: VBScript File Object OpenAsTextStream Method

The **OpenAsTextStream** method provides the same functionality as the **OpenTextFile** method of the **FileSystemObject**. In addition, the **OpenAsTextStream** method can be used to write to a file.

The following code illustrates the use of the **OpenAsTextStream** method:

```
Sub TextStreamTest

Const ForReading = 1, ForWriting = 2, ForAppending = 3

Const TristateUseDefault = -2, TristateTrue = -1, TristateFalse = 0

Dim fs, f, ts, s

Set fs = CreateObject("Scripting.FileSystemObject")
```

```
fs.CreateTextFile "test1.txt"
'Create a file
Set f = fs.GetFile("test1.txt")
Set ts = f.OpenAsTextStream(ForWriting, TristateUseDefault)
ts.Write "Hello World"
ts.Close
Set ts = f.OpenAsTextStream(ForReading, TristateUseDefault)
s = ts.ReadLine
Response.Write s
ts.Close
End Sub
```

VBScript File Object ParentFolder Property

Returns the folder object for the parent of the specified file. Read-only.

Syntax: VBScript File Object ParentFolder Property

```
object.ParentFolder
```

Arguments: VBScript File Object ParentFolder Property

object

A **File** object.

Remarks: VBScript File Object ParentFolder Property

The following code illustrates the use of the **ParentFolder** property with a file:

```
Sub ShowFileAccessInfo(filespec)
Dim fs, f, s
Set fs = CreateObject("Scripting.FileSystemObject")
Set f = fs.GetFile(filespec)
s = UCase(f.Name) & " in " & UCase(f.ParentFolder) & vbCrLf
s = s & "Created: " & f.DateCreated & vbCrLf
s = s & "Last Accessed: " & f.DateLastAccessed & vbCrLf
s = s & "Last Modified: " & f.DateLastModified
Response.Write s, 0, "File Access Info"
End Sub
```

VBScript File Object Path Property

Returns the path for a specified file.

Syntax: VBScript File Object Path Property

```
object.Path
```

Arguments: VBScript File Object Path Property

object

A **File** object.

Remarks: VBScript File Object Path Property

The following code illustrates the use of the **Path** property with a **File** object:

```
Sub ShowFileAccessInfo(filespec)
Dim fs, d, f, s
Set fs = CreateObject("Scripting.FileSystemObject")
Set f = fs.GetFile(filespec)
s = UCase(f.Path) & vbCrLf
s = s & "Created: " & f.DateCreated & vbCrLf
s = s & "Last Accessed: " & f.DateLastAccessed & vbCrLf
s = s & "Last Modified: " & f.DateLastModified
Response.Write s, 0, "File Access Info"
End Sub
```

VBScript File Object ShortName Property

Returns the short name used by programs that require the earlier 8.3 naming convention. *This property is not currently supported on UNIX.*

Syntax: VBScript File Object ShortName Property

```
object.ShortName
```

Arguments: VBScript File Object ShortName Property

object

A **File** object.

Remarks: VBScript File Object ShortName Property

The following code illustrates the use of the **ShortName** property with a **File** object:

```
Sub ShowShortName(filespec)
Dim fs, f, s
```

```
Set fs = CreateObject("Scripting.FileSystemObject")
Set f = fs.GetFile(filespec)
s = "The short name for " & "" & UCase(f.Name)
s = s & "" & vbCrLf
s = s & "is: " & "" & f.ShortName & ""
Response.Write s, 0, "Short Name Info"
End Sub
```

VBScript File Object ShortPath Property

Returns the short path used by programs that require the earlier 8.3 file naming convention. *This property is not currently supported on UNIX.*

Syntax: VBScript File Object ShortPath Property

```
object.ShortPath
```

Arguments: VBScript File Object ShortPath Property

object

A **File** object.

Remarks: VBScript File Object ShortPath Property

The following code illustrates the use of the **ShortPath** property with a **File** object:

```
Sub ShowShortPath(filespec)
Dim fs, f, s
Set fs = CreateObject("Scripting.FileSystemObject")
Set f = fs.GetFile(filespec)
s = "The short path for " & "" & UCase(f.Name)
s = s & "" & vbCrLf
s = s & "is: " & "" & f.ShortPath & ""
Response.Write s, 0, "Short Path Info"
End Sub
```

VBScript File Object Size Property

For files, returns the size, in bytes, of the specified file.

Syntax: VBScript File Object Size Property

```
object.Size
```

Arguments: VBScript File Object Size Property

object

A **File** object.

VBScript File Object Type Property

Returns information about the type of a file or folder. For example, for files ending in .TXT, "Text Document" is returned. *This property is not currently supported on UNIX.*

Syntax: VBScript File Object Type Property

```
object.Type
```

Arguments: VBScript File Object Type Property

object

A **File** object.

VBScript FileSystemObject Object

VBScript FileSystemObject Object

The VBScript **FileSystemObject** object provides access to a computer's file system.

Methods: VBScript FileSystemObject Object

VBScript FileSystemObject Object BuildPath Method	Appends a name to an existing path.
VBScript FileSystemObject Object CopyFile Method	Copies one or more files from one location to another.
VBScript FileSystemObject Object CopyFolder Method	Recursively copies a folder from one location to another.
VBScript FileSystemObject Object CreateFolder Method	Creates a folder.
VBScript FileSystemObject Object CreateTextFile Method	Creates a specified file name and returns a **TextStream** object.
VBScript FileSystemObject Object DeleteFile Method	Deletes one or more files.
VBScript FileSystemObject Object DeleteFolder Method	Deletes a folder and its contents.
VBScript FileSystemObject Object DriveExists Method	Indicates the existence of a drive.
VBScript FileSystemObject Object FileExists	Indicates the existence of a file.

Method	
VBScript FileSystemObject Object FolderExists Method	Indicates the existence of a folder.
VBScript FileSystemObject Object GetAbsolutepathname Method	Returns a complete and unambiguous path from a provided path specification.
VBScript FileSystemObject Object GetBaseName Method	Returns the base name of a path.
VBScript FileSystemObject Object GetDrive Method	Returns a **Drive** object corresponding to the drive in a path. *This method is not currently supported on UNIX.*
VBScript FileSystemObject Object GetDriveName Method	Returns a string containing the name of the drive for a path.
VBScript FileSystemObject Object GetExtensionName Method	Returns a string containing the extension for the last component in a path.
VBScript FileSystemObject Object GetFile Method	Returns a **File** object corresponding to the file in a path.
VBScript FileSystemObject Object GetFileName Method	Returns the last component of a path that is not part of the drive specification.
VBScript FileSystemObject Object GetFolder Method	Returns a **Folder** object corresponding to the folder in a specified path.
VBScript FileSystemObject Object GetParentFolderName Method	Returns a string containing the name of the parent folder of the last component in a path.
VBScript FileSystemObject Object GetSpecialFolder Method	Returns the special folder requested.
VBScript FileSystemObject Object GetTempName Method	Returns a randomly generated temporary file or folder name.
VBScript FileSystemObject Object MoveFile Method	Moves one or more files from one location to another.
VBScript FileSystemObject Object MoveFolder Method	Moves one or more folders from one location to another.
VBScript FileSystemObject Object OpenTextFile Method	Opens a file and returns a **TextStream** object.

Note

Collections returned by **FileSystemObject** method calls reflect the state of the file system when the collection was created. Changes to the file system after creation are not reflected in the collection. If the file system might be changed during the lifetime of the collection object, the method returning the collection should be called again to ensure that the contents are current.

Properties: FileSystemObject Object

VBScript FileSystemObject Object Drives Property	A **Drives** collection of all **Drive** objects available on the local machine.

Syntax: FileSystemObject Object

```
Scripting.FileSystemObject
```

Remarks: FileSystemObject Object

The following code illustrates how **FileSystemObject** is used to return a **TextStream** object that can be read from or written to:

Set fs = CreateObject("**Scripting.FileSystemObject**")

Set a = fs.CreateTextFile("c:\testfile.txt", True)

a.WriteLine("This is a test.")

a.Close

In the code shown above, the **CreateObject** function returns the **FileSystemObject** (`fs`). The **CreateTextFile** method then creates the file as a **TextStream** object (`a`) and the VBScript TextStream Object WriteLine Method writes a line of text to the created text file. The VBScript TextStream Object Close Method flushes the buffer and closes the file.

VBScript FileSystemObject Object BuildPath Method

Appends a name to an existing path.

Syntax: VBScript FileSystemObject Object BuildPath Method

```
object.BuildPath(path, name)
```

Arguments: VBScript FileSystemObject Object BuildPath Method

object

Always the name of a **FileSystemObject**. Required.

path

The existing *path* to which *name* is appended. *Path* can be absolute or relative and need not specify an existing folder. Required.

name

The name being appended to the existing *path*. Required.

Remarks: VBScript FileSystemObject Object BuildPath Method

The **BuildPath** method inserts an additional path separator between the existing *path* and the *name* if necessary.

VBScript FileSystemObject Object CopyFile Method

Copies one or more files from one location to another.

Syntax: VBScript FileSystemObject Object CopyFile Method

```
object.CopyFile source, destination[, overwrite]
```

Arguments: VBScript FileSystemObject Object CopyFile Method

object

The name of a **FileSystemObject**. Required.

source

The character string file specification, which can include wildcard characters, for one or more files to be copied. Required.

destination

The character string destination where the file or files from *source* are to be copied. Wildcard characters are not allowed. Required.

overwrite

A Boolean value that indicates if existing files are to be overwritten. If **True**, files are overwritten; if **False**, they are not. The default is **True**.

Note

> **CopyFile** will fail if *destination* has the read-only attribute set, regardless of the value of *overwrite*. Optional.

Remarks: VBScript FileSystemObject Object CopyFile Method

Wildcard characters can only be used in the last path component of the *source* argument. For example, you can use:

```
FileSystemObject.CopyFile "c:\mydocuments\letters\*.doc",
"c:\tempfolder\"

FileSystemObject.CopyFile "/home/letters/*.doc", "/var/tmp/"
```

But you can't use:

```
FileSystemObject.CopyFile "c:\mydocuments\*\R1???97.xls",
"c:\tempfolder"
```

If *source* contains wildcard characters or *destination* ends with a path separator (\), it is assumed that *destination* is an existing folder in which to copy matching files. Otherwise, *destination* is assumed to be the name of a file to create. In either case, three things can happen when an individual file is copied:

- If *destination* does not exist, *source* gets copied. This is the usual case.
- If *destination* is an existing file, an error occurs if overwrite is **False**. Otherwise, an attempt is made to copy source over the existing file.

- If *destination* is a directory, an error occurs.

An error also occurs if a *source* using wildcard characters doesn't match any files. The **CopyFile** method stops on the first error it encounters. No attempt is made to roll back or undo any changes made before an error occurs.

On UNIX systems symlinks are copied and a new symlink is created in the destination.

VBScript FileSystemObject Object CopyFolder Method

Recursively copies a folder from one location to another.

Syntax: VBScript FileSystemObject Object CopyFolder Method

```
object.CopyFolder source, destination[, overwrite]
```

Arguments: VBScript FileSystemObject Object CopyFolder Method

object

The name of a **FileSystemObject**. Required.

source

The character string folder specification, which can include wildcard characters, for one or more folders to be copied. Required.

destination

The character string *destination* where the folder and subfolders from *source* are to be copied. Wildcard characters are not allowed. Required.

overwrite

A Boolean value that indicates if existing folders are to be overwritten. If **True**, files are overwritten; if **False**, they are not. The default is **True**. Optional.

Remarks: VBScript FileSystemObject Object CopyFolder Method

Wildcard characters can only be used in the last path component of the source argument. For example, you can use:

```
FileSystemObject.CopyFolder "c:\mydocuments\letters\*",
"c:\tempfolder\"
```

But you can't use:

```
FileSystemObject.CopyFolder "c:\mydocuments\*\*", "c:\tempfolder\"
```

If *source* contains wildcard characters or *destination* ends with a path separator (\), it is assumed that *destination* is an existing folder in which to copy matching folders and subfolders. Otherwise, *destination* is assumed to be the name of a folder to create. In either case, four things can happen when an individual folder is copied:

If *destination* does not exist, the source folder and all its contents gets copied. This is the usual case.

- If *destination* is an existing file, an error occurs.
- If *destination* is a directory, an attempt is made to copy the folder and all its contents. If a file contained in *source* already exists in *destination,* an error occurs if overwrite is **False**. Otherwise, it will attempt to copy the file over the existing file.
- If *destination* is a read-only directory, an error occurs if an attempt is made to copy an existing read-only file into that directory and overwrite is **False**.
- An error also occurs if a *source* using wildcard characters doesn't match any folders.

The **CopyFolder** method stops on the first error it encounters. No attempt is made to roll back any changes made before an error occurs.

On UNIX systems, symlinks are copied and a new symlink is created in the destination.

VBScript FileSystemObject Object CreateFolder Method

Creates a folder.

Syntax: VBScript FileSystemObject Object CreateFolder Method

```
object.CreateFolder(foldername)
```

Arguments: VBScript FileSystemObject Object CreateFolder Method

object

The name of a **FileSystemObject**. Required.

foldername

A string expression that identifies the folder to create. Required.

Remarks: VBScript FileSystemObject Object CreateFolder Method

An error occurs if the specified folder already exists.

VBScript FileSystemObject Object CreateTextFile Method

Creates a specified file name and returns a **TextStream** object that can be used to read from or write to the file.

Syntax: VBScript FileSystemObject Object CreateTextFile Method

```
object.CreateTextFile(filename[, overwrite[, unicode]])
```

Arguments: VBScript FileSystemObject Object CreateTextFile Method

object

The name of a **FileSystemObject** or **Folder** object. Required.

filename

A string expression that identifies the file to create. Required.

overwrite

A Boolean value that indicates if an existing file can be overwritten. The value is **True** if the file can be overwritten; **False** if it can't be overwritten. If omitted, existing files are not overwritten. Optional.

unicode

A Boolean value that indicates whether the file is created as a Unicode or ASCII file. The value is **True** if the file is created as a Unicode file; **False** if it's created as an ASCII file. If omitted, an ASCII file is assumed. Optional.

Remarks: VBScript FileSystemObject Object CreateTextFile Method

The following code illustrates how to use the **CreateTextFile** method to create and open a text file:

```
Sub CreateAfile

Set fs = CreateObject("Scripting.FileSystemObject")

Set a = fs.CreateTextFile("c:\testfile.txt", True)

a.WriteLine("This is a test.")

a.Close

End Sub
```

If the *overwrite* argument is **False**, or is not provided, for a *filename* that already exists, an error occurs.

VBScript FileSystemObject Object DeleteFile Method

Deletes a specified file.

Syntax: VBScript FileSystemObject Object DeleteFile Method

```
object.DeleteFile filespec[, force]
```

Arguments: VBScript FileSystemObject Object DeleteFile Method

object

The name of a **FileSystemObject**. Required.

filespec

The name of the file to delete. The *filespec* can contain wildcard characters in the last path component. Required.

force

A Boolean value that is **True** if files with the read-only attribute set are to be deleted; **False** (default) if they are not. Optional.

Remarks: VBScript FileSystemObject Object DeleteFile Method

An error occurs if no matching files are found. The **DeleteFile** method stops on the first error it encounters. No attempt is made to roll back or undo any changes that were made before an error occurred.

VBScript FileSystemObject Object DeleteFolder Method

Deletes a specified folder and its contents.

Syntax: VBScript FileSystemObject Object DeleteFolder Method

```
object.DeleteFolder folderspec[, force]
```

Arguments: VBScript FileSystemObject Object DeleteFolder Method

object

The name of a **FileSystemObject**. Required.

folderspec

The name of the folder to delete. The *folderspec* can contain wildcard characters in the last path component. Required.

force

A Boolean value that is **True** if folders with the read-only attribute set are to be deleted; **False** (default) if they are not. Optional.

Remarks: VBScript FileSystemObject Object DeleteFolder Method

The **DeleteFolder** method does not distinguish between folders that have contents and those that do not. The specified folder is deleted regardless of whether or not it has contents.

An error occurs if no matching folders are found. The **DeleteFolder** method stops on the first error it encounters. No attempt is made to roll back or undo any changes that were made before an error occurred.

VBScript FileSystemObject Object DriveExists Method

Returns **True** if the specified drive exists; **False** if it does not.

Syntax: VBScript FileSystemObject Object DriveExists Method

```
object.DriveExists(drivespec)
```

Arguments: VBScript FileSystemObject Object DriveExists Method

object

The name of a **FileSystemObject**. Required.

drivespec

A drive letter or a complete path specification. Required.

Remarks: VBScript FileSystemObject Object DriveExists Method

For drives with removable media, the **DriveExists** method returns **True** even if there are no media present. Use the VBScript Drive Object IsReady Property of the **Drive** object to determine if a drive is ready.

On UNIX systems the only valid drive name is "/".

VBScript FileSystemObject Object Drives Property

Returns a **Drives** collection consisting of all **Drive** objects available on the local machine.

Syntax: VBScript FileSystemObject Object Drives Property

object.**Drives**

Arguments: VBScript FileSystemObject Object Drives Property

object

A **FileSystemObject**.

Remarks: VBScript FileSystemObject Object Drives Property

Removable-media drives need not have media inserted for them to appear in the **Drives** collection.

Under UNIX the **Drives** collection contains only one member, "/".

You can iterate the members of the **Drives** collection using a **For Each…Next** construct as illustrated in the following code:

```
Sub ShowDriveList

Dim fs, d, dc, s, n

Set fs = CreateObject("Scripting.FileSystemObject")

Set dc = fs.Drives

For Each d in dc

s = s & d.DriveLetter & " - "

If d.DriveType = 3 Then

n = d.ShareName

Else

n = d.VolumeName

End If

s = s & n & vbCrLf

Next

Response.Write s

End Sub
```

VBScript FileSystemObject Object FileExists Method

Returns **True** if a specified file exists; **False** if it does not.

Syntax: VBScript FileSystemObject Object FileExists Method

```
object.FileExists(filespec)
```

Arguments: VBScript FileSystemObject Object FileExists Method

object

The name of a **FileSystemObject**. Required.

filespec

The name of the file whose existence is to be determined. A complete path specification (either absolute or relative) must be provided if the file isn't expected to exist in the current folder. Required.

VBScript FileSystemObject Object FolderExists Method

Returns **True** if a specified folder exists; **False** if it does not.

Syntax: VBScript FileSystemObject Object FolderExists Method

```
object.FolderExists(folderspec)
```

Arguments: VBScript FileSystemObject Object FolderExists Method

object

Required. Always the name of a **FileSystemObject**.

folderspec

Required. The name of the folder whose existence is to be determined. A complete path specification (either absolute or relative) must be provided if the folder isn't expected to exist in the current folder.

VBScript FileSystemObject Object GetAbsolutepathname Method

Returns a complete and unambiguous path from a provided path specification.

Syntax: VBScript FileSystemObject Object GetAbsolutepathname Method

```
object.GetAbsolutepathname(pathspec)
```

Arguments: VBScript FileSystemObject Object GetAbsolutepathname Method

object

The name of a **FileSystemObject**. Required.

pathspec

The path specification to change to a complete and unambiguous path. Required.

Remarks: VBScript FileSystemObject Object GetAbsolutepathname Method

A path is complete and unambiguous if it provides a complete reference from the root of the specified drive. A complete path can only end with a path separator character (\) if it specifies the root folder of a mapped drive.

Assuming the current directory is c:\mydocuments\reports, the following table illustrates the behavior of the **GetAbsolutepathname** method.

pathspec	Returned path
"c:"	"c:\mydocuments\reports"
"c:.."	"c:\mydocuments"
"c:\\\"	"c:\"
"c:*.*\may97"	"c:\mydocuments\reports*.*\may97"
"region1"	"c:\mydocuments\reports\region1"
"c:\..\..\mydocuments"	"c:\mydocuments"

VBScript FileSystemObject Object GetBaseName Method

Returns a string containing the base name of the last component, less any file extension, in a path.

Syntax: VBScript FileSystemObject Object GetBaseName Method

```
object.GetBaseName(path)
```

Arguments: VBScript FileSystemObject Object GetBaseName Method

object

The name of a **FileSystemObject**. Required.

path

The path specification for the component whose base name is to be returned. Required.

Remarks: VBScript FileSystemObject Object GetBaseName Method

The **GetBaseName** method returns a zero-length string ("") if no component matches the *path* argument.

Note

> The **GetBaseName** method works only on the provided *path* string. It does not attempt to resolve the path, nor does it check for the existence of the specified path.

VBScript FileSystemObject Object GetDrive Method

Returns a **Drive** object corresponding to the drive in a specified path.

Syntax: VBScript FileSystemObject Object GetDrive Method

```
object.GetDrive drivespec
```

Arguments: VBScript FileSystemObject Object GetDrive Method

object

The name of a **FileSystemObject**. Required.

drivespec

The *drivespec* argument. It can be a drive letter (c), a drive letter with a colon appended (c:), a drive letter with a colon and path separator appended (c:\), or any network share specification (\\computer2\share1). Required.

Remarks: VBScript FileSystemObject Object GetDrive Method

For network shares, a check is made to ensure that the share exists.

An error occurs if *drivespec* does not conform to one of the accepted forms or does not exist.

To call the **GetDrive** method on a normal path string, use the following sequence to get a string that is suitable for use as *drivespec:*

```
DriveSpec = GetDriveName(GetAbsolutepathname(Path))
```

On UNIX systems the **GetDrive** method will only return a **Drive** object for "/".

VBScript FileSystemObject Object GetDriveName Method

Returns a string containing the name of the drive for a specified path.

Syntax: VBScript FileSystemObject Object GetDriveName Method

```
object.GetDriveName(path)
```

Arguments: VBScript FileSystemObject Object GetDriveName Method

object

The name of a **FileSystemObject**. Required.

path

The path specification for the component whose drive name is to be returned. Required.

Remarks: VBScript FileSystemObject Object GetDriveName Method

The **GetDriveName** method returns a zero-length string ("") if the drive can't be determined. On UNIX systems the **GetDriveName** method returns "/".

Note

The **GetDriveName** method works only on the provided *path* string. It does not attempt to resolve the path, nor does it check for the existence of the specified path.

VBScript FileSystemObject Object GetExtensionName Method

Returns a string containing the extension name for the last component in a path.

Syntax: VBScript FileSystemObject Object GetExtensionName Method

```
object.GetExtensionName(path)
```

Arguments: VBScript FileSystemObject Object GetExtensionName Method

object

The name of a **FileSystemObject**. Required.

path

The path specification for the component whose extension name is to be returned. Required.

Remarks: VBScript FileSystemObject Object GetExtensionName Method

For network drives, the root directory (\) is considered to be a component.

The **GetExtensionName** method returns a zero-length string ("") if no component matches the *path* argument.

VBScript FileSystemObject Object GetFile Method

Returns a **File** object corresponding to the file in a specified path.

Syntax: VBScript FileSystemObject Object GetFile Method

```
object.GetFile(filespec)
```

Arguments: VBScript FileSystemObject Object GetFile Method

object

The name of a **FileSystemObject**. Required.

filespec

The **filespec** is the path (absolute or relative) to a specific file. Required.

Remarks: VBScript FileSystemObject Object GetFile Method

An error occurs if the specified file does not exist.

VBScript FileSystemObject Object GetFileName Method

Returns the last component of a specified path that is not part of the drive specification.

Syntax: VBScript FileSystemObject Object GetFileName Method

```
object.GetFileName(pathspec)
```

Arguments: VBScript FileSystemObject Object GetFileName Method

object

The name of a **FileSystemObject**. Required.

pathspec

The path (absolute or relative) to a specific file. Required.

Remarks: VBScript FileSystemObject Object GetFileName Method

The **GetFileName** method returns a zero-length string ("") if *pathspec* does not end with the named component.

Note

> The **GetFileName** method works only on the provided *path* string. It does not attempt to resolve the path, nor does it check for the existence of the specified path.

VBScript FileSystemObject Object GetFolder Method

Returns a Folder object corresponding to the folder in a specified path.

Syntax: VBScript FileSystemObject Object GetFolder Method

```
object.GetFolder(folderspec)
```

Arguments: VBScript FileSystemObject Object GetFolder Method

object

The name of a **FileSystemObject**. Required.

folderspec

The path (absolute or relative) to a specific folder. Required.

Remarks: VBScript FileSystemObject Object GetFolder Method

An error occurs if the specified folder does not exist.

VBScript FileSystemObject Object GetParentFolderName Method

Returns a string containing the name of the parent folder of the last component in a specified path.

Syntax: VBScript FileSystemObject Object GetParentFolderName Method

```
object.GetParentFolderName(path)
```

Arguments: VBScript FileSystemObject Object GetParentFolderName Method

object

The name of a **FileSystemObject**. Required.

path

The path specification for the component whose parent folder name is to be returned. Required.

Remarks: VBScript FileSystemObject Object GetParentFolderName Method

The **GetParentFolderName** method returns a zero-length string ("") if there is no parent folder for the component specified in the *path* argument.

Note

> The **GetParentFolderName** method works only on the provided *path* string. It does not attempt to resolve the path, nor does it check for the existence of the specified path.

VBScript FileSystemObject Object GetSpecialFolder Method

Returns the special folder specified.

Syntax: VBScript FileSystemObject Object GetSpecialFolder Method

```
object.GetSpecialFolder(folderspec)
```

Arguments: VBScript FileSystemObject Object GetSpecialFolder Method

object

The name of a **FileSystemObject**. Required.

folderspec

The name of the special folder to be returned. Can be any of the constants shown in the Settings section. Required.

Settings: VBScript FileSystemObject Object GetSpecialFolder Method

The *folderspec* argument can have any of the following values:

Constant	Value	Description
WindowsFolder	0	The Windows folder contains files installed by the Windows operating system. *Not supported on Unix.*
SystemFolder	1	The System folder contains libraries, fonts, and device drivers. *Not supported on Unix.*
TemporaryFolder	2	The Temp folder is used to store temporary files. Its path is found in the TMP environment variable.

VBScript FileSystemObject Object GetTempName Method

Returns a randomly generated temporary file or folder name that is useful for performing operations that require a temporary file or folder.

Syntax: VBScript FileSystemObject Object GetTempName Method

```
object.GetTempName
```

Arguments: VBScript FileSystemObject Object GetTempName Method

object

An optional argument that is always the name of a **FileSystemObject**.

Remarks: VBScript FileSystemObject Object GetTempName Method

The **GetTempName** method does not create a file. It provides only a temporary file name that can be used with the **CreateTextFile** method to create a file.

VBScript FileSystemObject Object MoveFile Method

Moves one or more files from one location to another.

Syntax: VBScript FileSystemObject Object MoveFile Method

```
object.MoveFile source, destination
```

Arguments: VBScript FileSystemObject Object MoveFile Method

object

The name of a **FileSystemObject**. Required.

source

The path to the file or files to be moved. The *source* argument string can contain wildcard characters in the last path component only. Required.

destination

The path where the file or files are to be moved. The *destination* argument can't contain wildcard characters. Required.

Remarks: VBScript FileSystemObject Object MoveFile Method

If *source* contains wildcards or *destination* ends with a path separator (\), it is assumed that *destination* specifies an existing folder in which to move the matching files. Otherwise, *destination* is assumed to be the name of a destination file to create. In either case, three things can happen when an individual file is moved:

- If *destination* does not exist, the file is moved. This is the usual case.
- If *destination* is an existing file, an error occurs.
- If *destination* is a directory, an error occurs.

An error also occurs if a wildcard character that is used in *source* doesn't match any files. The **MoveFile** method stops on the first error it encounters. No attempt is made to roll back any changes made before the error occurs.

On UNIX systems, symlinks are copied and a new symlink is created in the destination.

Note

This method allows moving files between volumes only if supported by the operating system.

VBScript FileSystemObject Object MoveFolder Method

Moves one or more folders from one location to another.

Syntax: VBScript FileSystemObject Object MoveFolder Method

```
object.MoveFolder source, destination
```

Arguments: VBScript FileSystemObject Object MoveFolder Method

object

Always the name of a **FileSystemObject**. Required.

source

The path to the folder or folders to be moved. The *source* arguments string can contain wildcard characters in the last path component only. Required.

destination

The path where the folder or folders are to be moved. The *destination* arguments can't contain wildcard characters. Required.

Remarks: VBScript FileSystemObject Object MoveFolder Method

If *source* contains wildcards or *destination* ends with a path separator (\), it is assumed that *destination* specifies an existing folder in which to move the matching files. Otherwise, *destination* is assumed to be the name of a destination folder to create. In either case, three things can happen when an individual folder is moved:

- If *destination* does not exist, the folder gets moved. This is the usual case.
- If *destination* is an existing file, an error occurs.
- If *destination* is a directory, an error occurs.

An error also occurs if a wildcard character that is used in *source* doesn't match any folders. The **MoveFolder** method stops on the first error it encounters. No attempt is made to roll back any changes made before the error occurs.

On UNIX systems, symlinks are copied and a new symlink is created in the destination.

Note

This method allows moving folders between volumes only if supported by the operating system.

VBScript FileSystemObject Object OpenTextFile Method

Opens a specified file and returns a **TextStream** object that can be used to read from or append to the file.

Syntax: VBScript FileSystemObject Object OpenTextFile Method

```
object.OpenTextFile(filename[, iomode[, create[, format]]])
```

Arguments: VBScript FileSystemObject Object OpenTextFile Method

object

The name of a **FileSystemObject**. Required.

filename

A string expression that identifies the file to open. Required.

iomode

Indicates input/output mode. Can be one of three constants: **ForReading, ForWriting,** or **ForAppending**. Optional.

create

A Boolean value that indicates whether a new file can be created if the specified filename doesn't exist. The value is **True** if a new file is created; **False** if it isn't created. The default is **False**. Optional.

format

One of three **Tristate** values used to indicate the format of the opened file. If omitted, the file is opened as ASCII. Optional.

Settings: VBScript FileSystemObject Object OpenTextFile Method

The *iomode* arguments can have any of the following settings:

Constant	Value	Description
ForReading	1	Open a file for reading only. You can't write to this file.
ForWriting	2	Open a file for writing. If a file with the same name exists, its previous contents are overwritten.
ForAppending	8	Open a file and write to the end of the file.

The *format* argument can have any of the following settings:

Constant	Value	Description
TristateUseDefault	-2	Opens the file using the system default.
TristateTrue	-1	Opens the file as Unicode.
TristateFalse	0	Opens the file as ASCII.

Remarks: VBScript FileSystemObject Object OpenTextFile Method

The following code illustrates the use of the **OpenTextFile** method to open a file for appending text:

```
Sub OpenTextFileTest

Const ForReading = 1, ForWriting = 2, ForAppending = 3

Dim fs, f

Set fs = CreateObject("Scripting.FileSystemObject")

Set f = fs.OpenTextFile("c:\testfile.txt",
ForAppending,TristateFalse)

f.Write "Hello world!"

f.Close

End Sub
```

VBScript Folder Object

VBScript Folder Object

The VBScript **Folder** object provides access to all the properties of a folder.

Methods: VBScript Folder Object

VBScript Folder Object Copy Method	Copies a folder from one location to another.
VBScript Folder Object Delete Method	Deletes a folder.
VBScript Folder Object Move Method	Moves a folder from one location to another.
VBScript Folder Object CreateTextFile Method	Creates a file and returns a **TextStream** object.

Property: VBScript Folder Object

VBScript Folder Object Attributes Property	The attributes of a folder.
VBScript Folder Object DateCreated Property	The date and time a folder was created.
VBScript Folder Object DateLastAccessed Property	The date and time that the folder was last accessed.
VBScript Folder Object DateLastModified Property	The date and time that the folder was last modified.
VBScript Folder Object Drive Property	The drive letter of the drive on which the folder resides.
VBScript Folder Object Files Property	A **Files** collection of all **File** objects in the folder.
VBScript Folder Object IsRootFolder Property	**True** if this is the root folder of a drive.

VBScript Folder Object Name Property	The name of the folder.
VBScript Folder Object ParentFolder Property	The **Folder** object for the parent of the folder.
VBScript Folder Object Path Property	The file system path to the folder.
VBScript Folder Object ShortName Property	The short name used by programs that require 8.3 names. *This property is not currently supported on UNIX.*
VBScript Folder Object ShortPath Property	The short path used by programs that require 8.3 names. *This property is not currently supported on UNIX.*
VBScript Folder Object Size Property	The size, in bytes, of all files and subfolders contained in a folder.
VBScript Folder Object SubFolders Property	A **Folders** collection containing all the folders in a **Folder** object.

Remarks: VBScript Folder Object

The following code illustrates how to obtain a **Folder** object and how to return one of its properties:

Sub ShowFolderInfo(folderspec)

Dim fs, f, s,

Set fs = CreateObject("Scripting.FileSystemObject")

Set **f** = fs.GetFolder(folderspec)

s = **f.**DateCreated

Response.Write s

End Sub

VBScript Folder Object Attributes Property

Sets or returns the attributes of folders. Read/write or read-only, depending on the attribute.

Note

This property depends on the underlying operating system for its behavior. If the OS file system does not support the folder attribute requested, an error will be returned.

Syntax: VBScript Folder Object Attributes Property

```
object.Attributes [= newattributes]
```

Arguments: VBScript Folder Object Attributes Property

object

The name of a **Folder** object. Required.

newattributes

The new value for the attributes of the specified object. Optional.

Settings: VBScript Folder Object Attributes Property

The *newattributes* argument can have any of the following values or any logical combination of the following values:

Constant	Value	Description
Normal	0	Normal file. No attributes are set.
ReadOnly	1	Read-only file. Attribute is read/write.
Hidden	2	Hidden file. Attribute is read/write.
System	4	System file. Attribute is read/write.
Volume	8	Disk drive volume label. Attribute is read-only.
Directory	16	Folder or directory. Attribute is read-only.
Archive	32	File has changed since last backup. Attribute is read/write.
Alias	64	Link or shortcut. Attribute is read-only.
Compressed	128	Compressed file. Attribute is read-only.

VBScript Folder Object Copy Method

Copies a specified folder from one location to another.

Syntax: VBScript Folder Object Copy Method

```
object.Copy destination[, overwrite]
```

Arguments: VBScript Folder Object Copy Method

object

The name of a **Folder** object. Required.

destination

The destination where the folder is to be copied. Wildcard characters are not allowed. Required.

overwrite

A Boolean value that is **True** (default) if existing folders are to be overwritten; **False** if they are not. Optional.

Remarks: VBScript Folder Object Copy Method

The results of the **Copy** method on a **Folder** object are identical to operations performed using **CopyFolder** where the file or folder referred to by *object* is passed as an argument. You should note, however, that the alternative method is capable of copying multiple files or folders.

VBScript Folder Object CreateTextFile Method

Creates a specified file name and returns a **TextStream** object that can be used to read from or write to the file.

Syntax: VBScript Folder Object CreateTextFile Method

```
object.CreateTextFile(filename[, overwrite[, unicode]])
```

Arguments: VBScript Folder Object CreateTextFile Method

object

The name of a **Folder** object. Required.

filename

A string expression that identifies the file to create. Required.

overwrite

A Boolean value that indicates if an existing file can be overwritten. The value is **True** if the file can be overwritten; **False** if it can't be overwritten. If omitted, existing files are not overwritten. Optional.

unicode

A Boolean value that indicates whether the file is created as a Unicode or ASCII file. The value is **True** if the file is created as a Unicode file; **False** if it's created as an ASCII file. If omitted, an ASCII file is assumed. Optional.

Remarks: VBScript Folder Object CreateTextFile Method

The following code illustrates how to use the **CreateTextFile** method to create and open a text file:

```
Sub CreateAfile

Set fd = CreateObject("Scripting.Folder")

Set a = fd.CreateTextFile("c:\testfile.txt", True)

a.WriteLine("This is a test.")

a.Close

End Sub
```

If the *overwrite* argument is **False**, or is not provided for a filename that already exists, an error occurs.

VBScript Folder Object DateCreated Property

Returns the date and time that the specified folder was created. Read-only.

Syntax: VBScript Folder Object DateCreated Property

```
object.DateCreated
```

Arguments: VBScript Folder Object DateCreated Property

object

A **Folder** object.

Remarks: VBScript Folder Object DateCreated Property

The following code illustrates the use of the **DateCreated** property with a folder:

```
Sub ShowFileInfo(filespec)
Dim fs, f, s
Set fs = CreateObject("Scripting.FileSystemObject")
Set f = fs.GetFolder(filespec)
s = "Created: " & f.DateCreated
Response.Write s
End Sub
```

VBScript Folder Object DateLastAccessed Property

Returns the date and time that the specified folder was last accessed. Read-only.

Syntax: VBScript Folder Object DateLastAccessed Property

```
object.DateLastAccessed
```

Arguments: VBScript Folder Object DateLastAccessed Property

object

A **Folder** object.

Remarks: VBScript Folder Object DateLastAccessed Property

The following code illustrates the use of the **DateLastAccessed** property with a folder:

```
Sub ShowFileAccessInfo(filespec)
Dim fs, f, s
Set fs = CreateObject("Scripting.FileSystemObject")
Set f = fs.GetFolder(filespec)
s = UCase(filespec) & vbCrLf
s = s & "Created: " & f.DateCreated & vbCrLf
```

```
s = s & "Last Accessed: " & f.DateLastAccessed & vbCrLf
s = s & "Last Modified: " & f.DateLastModified
Response.Write s, 0, "Folder Access Info"
End Sub
```

Note

This method depends on the underlying operating system for its behavior. If the operating system does not support providing time information, none will be returned.

VBScript Folder Object DateLastModified Property

Returns the date and time that the specified folder was last modified. Read-only.

Syntax: VBScript Folder Object DateLastModified Property

```
object.DateLastModified
```

Arguments: VBScript Folder Object DateLastModified Property

object

A **Folder** object.

Remarks: VBScript Folder Object DateLastModified Property

The following code illustrates the use of the **DateLastModified** property with a folder:

```
Sub ShowFileAccessInfo(filespec)
Dim fs, f, s
Set fs = CreateObject("Scripting.FileSystemObject")
Set f = fs.GetFolder(filespec)
s = UCase(filespec) & vbCrLf
s = s & "Created: " & f.DateCreated & vbCrLf
s = s & "Last Accessed: " & f.DateLastAccessed & vbCrLf
s = s & "Last Modified: " & f.DateLastModified
Response.Write s, 0, "Folder Access Info"
End Sub
```

VBScript Folder Object Delete Method

Deletes a specified folder.

Syntax: VBScript Folder Object Delete Method

```
object.Delete force
```

Arguments: VBScript Folder Object Delete Method

object

The name of a **Folder** object. Required.

force

A Boolean value that is **True** if folders with the read-only attribute set are to be deleted; **False** (default) if they are not. Optional.

Remarks: VBScript Folder Object Delete Method

An error occurs if the specified folder does not exist.

The results of the **Delete** method on a **Folder** are identical to operations performed using **DeleteFolder**.

The **Delete** method does not distinguish between folders that have contents and those that do not. The specified folder is deleted regardless of whether or not it has contents.

VBScript Folder Object Drive Property

Returns the drive letter of the drive on which the specified folder resides. Read-only.

Syntax: VBScript Folder Object Drive Property

```
object.Drive
```

Arguments: VBScript Folder Object Drive Property

object

A **Folder** object.

Remarks: VBScript Folder Object Drive Property

The following code illustrates the use of the **Drive** property:

```
Sub ShowFileAccessInfo(filespec)
Dim fs, f, s
Set fs = CreateObject("Scripting.FileSystemObject")
Set f = fs.GetFolder(filespec)
s = f.Name & " on Drive " & UCase(f.Drive) & vbCrLf
s = s & "Created: " & f.DateCreated & vbCrLf
s = s & "Last Accessed: " & f.DateLastAccessed & vbCrLf
s = s & "Last Modified: " & f.DateLastModified
Response.Write s, 0, "Folder Access Info"
End Sub
```

Under UNIX, the **Drive** property is always "/".

VBScript Folder Object Files Property

Returns a VBScript Files Collection consisting of all **File** objects contained in the specified folder, including those with hidden and system file attributes set.

Syntax: VBScript Folder Object Files Property

```
object.Files
```

Arguments: VBScript Folder Object Files Property

object

A **Folder** object.

Remarks: VBScript Folder Object Files Property

The following code illustrates the use of the **Files** property:

```
Sub ShowFileList(folderspec)
Dim fs, f, f1, fc, s
Set fs = CreateObject("Scripting.FileSystemObject")
Set f = fs.GetFolder(folderspec)
Set fc = f.Files
For Each f1 in fc
s = s & f1.name
s = s & vbCrLf
Next
Response.Write s
End Sub
```

VBScript Folder Object IsRootFolder Property

Returns **True** if the specified folder is the root folder; **False** if it is not.

Syntax: VBScript Folder Object IsRootFolder Property

```
object.IsRootFolder
```

Arguments: VBScript Folder Object IsRootFolder Property

object

A **Folder** object.

Remarks: VBScript Folder Object IsRootFolder Property

The following code illustrates the use of the **IsRootFolder** property:

```
Dim fs
Set fs = CreateObject("Scripting.FileSystemObject")
Sub DisplayLevelDepth(pathspec)
Dim f, n
Set f = fs.GetFolder(pathspec)
If f.IsRootFolder Then
MsgBox "The specified folder is the root folder."
Else
Do Until f.IsRootFolder
Set f = f.ParentFolder
n = n + 1
Loop
MsgBox "The specified folder is nested " & n & " levels deep."
End If
End Sub
```

VBScript Folder Object Move Method

Moves a specified folder from one location to another.

Syntax: VBScript Folder Object Move Method

```
object.Move destination
```

Arguments: VBScript Folder Object Move Method

object

Required. Always the name of a **Folder** object.

destination

Required. Destination where the folder is to be moved. Wildcard characters are not allowed.

Remarks: VBScript Folder Object Move Method

The results of the **Move** method on a **Folder** are identical to operations performed using **MoveFolder**. You should note, however, that the alternative method is capable of moving multiple files or folders.

VBScript Folder Object Name Property

Sets or returns the name of a specified folder. Read/write.

Syntax: VBScript Folder Object Name Property

```
object.Name [= newname]
```

Arguments: VBScript Folder Object Name Property

object

The name of a **Folder** object. Required.

newname

If provided, *newname* is the new name of the specified object. Optional.

Remarks: VBScript Folder Object Name Property

The following code illustrates the use of the **Name** property:

```
Sub ShowFileAccessInfo(filespec)
Dim fs, f, s
Set fs = CreateObject("Scripting.FileSystemObject")
Set f = fs.GetFolder(filespec)
s = f.Name & " on Drive " & UCase(f.Drive) & vbCrLf
s = s & "Created: " & f.DateCreated & vbCrLf
s = s & "Last Accessed: " & f.DateLastAccessed & vbCrLf
s = s & "Last Modified: " & f.DateLastModified
Response.Write s, 0, "Folder Access Info"
End Sub
```

VBScript Folder Object ParentFolder Property

Returns the folder object for the parent of the specified folder. Read-only.

Syntax: VBScript Folder Object ParentFolder Property

```
object.ParentFolder
```

Arguments: VBScript Folder Object ParentFolder Property

object

A **Folder** object.

Remarks: VBScript Folder Object ParentFolder Property

The following code illustrates the use of the **ParentFolder** property with a folder:

```
Sub ShowFileAccessInfo(filespec)
Dim fs, f, s
Set fs = CreateObject("Scripting.FileSystemObject")
```

```
Set f = fs.GetFolder(filespec)
s = UCase(f.Name) & " in " & UCase(f.ParentFolder) & vbCrLf
s = s & "Created: " & f.DateCreated & vbCrLf
s = s & "Last Accessed: " & f.DateLastAccessed & vbCrLf
s = s & "Last Modified: " & f.DateLastModified
Response.Write s, 0, "Folder Access Info"
End Sub
```

VBScript Folder Object Path Property

Returns the path for a specified folder.

Syntax: VBScript Folder Object Path Property

```
object.Path
```

Arguments: VBScript Folder Object Path Property

object

A **Folder** object.

Remarks: VBScript Folder Object Path Property

The following code illustrates the use of the **Path** property with a **Folder** object:

```
Sub ShowFileAccessInfo(filespec)
Dim fs, d, f, s
Set fs = CreateObject("Scripting.FileSystemObject")
Set f = fs.GetFolder(filespec)
s = UCase(f.Path) & vbCrLf
s = s & "Created: " & f.DateCreated & vbCrLf
s = s & "Last Accessed: " & f.DateLastAccessed & vbCrLf
s = s & "Last Modified: " & f.DateLastModified
Response.Write s, 0, "Folder Access Info"
End Sub
```

VBScript Folder Object ShortName Property

Returns the short name used by programs that require the earlier 8.3 naming convention. *This property is not currently supported on UNIX.*

Syntax: VBScript Folder Object ShortName Property

```
object.ShortName
```

Arguments: VBScript Folder Object ShortName Property

object

A **Folder** object.

Remarks: VBScript Folder Object ShortName Property

The following code illustrates the use of the **ShortName** property with a **Folder** object:

```
Sub ShowShortName(filespec)
Dim fs, f, s
Set fs = CreateObject("Scripting.FileSystemObject")
Set f = fs.GetFfolder(filespec)
s = "The short name for " & "" & UCase(f.Name)
s = s & "" & vbCrLf
s = s & "is: " & "" & f.ShortName & ""
Response.Write s, 0, "Short Name Info"
End Sub
```

VBScript Folder Object ShortPath Property

Returns the short path used by programs that require the earlier 8.3 file naming convention. *This property is not currently supported on UNIX.*

Syntax: VBScript Folder Object ShortPath Property

```
object.ShortPath
```

Arguments: VBScript Folder Object ShortPath Property

object

A **Folder** object.

Remarks: VBScript Folder Object ShortPath Property

The following code illustrates the use of the **ShortName** property with a **Folder** object:

```
Sub ShowShortPath(filespec)
Dim fs, f, s
Set fs = CreateObject("Scripting.FileSystemObject")
Set f = fs.GetFolder(filespec)
s = "The short path for " & "" & UCase(f.Name)
s = s & "" & vbCrLf
```

```
s = s & "is: " & "" & f.ShortPath & ""
Response.Write s, 0, "Short Path Info"
End Sub
```

VBScript Folder Object Size Property

Returns the size, in bytes, of all files and subfolders contained in the folder.

Syntax: VBScript Folder Object Size Property

```
object.Size
```

Arguments: VBScript Folder Object Size Property

object

A **Folder** object.

Remarks: VBScript Folder Object Size Property

The following code illustrates the use of the **Size** property with a **Folder** object:

```
Sub ShowFolderSize(filespec)
Dim fs, f, s
Set fs = CreateObject("Scripting.FileSystemObject")
Set f = fs.GetFolder(filespec)
s = UCase(f.Name) & " uses " & f.size & " bytes."
Response.Write s, 0, "Folder Size Info"
End Sub
```

VBScript Folder Object SubFolders Property

Returns a **Folders** collection consisting of all folders contained in a specified folder, including those with Hidden and System file attributes set.

Syntax: VBScript Folder Object SubFolders Property

```
object.SubFolders
```

Arguments: VBScript Folder Object SubFolders Property

object

A **Folder** object.

Remarks: VBScript Folder Object SubFolders Property

The following code illustrates the use of the **SubFolders** property:

```
Sub ShowFolderList(folderspec)
```

```
Dim fs, f, f1, s, sf
Set fs = CreateObject("Scripting.FileSystemObject")
Set f = fs.GetFolder(folderspec)
Set sf = f.SubFolders
For Each f1 in sf
s = s & f1.name
s = s & vbCrLf
Next
Response.Write s
End Sub
```

VBScript TextStream Object

VBScript TextStream Object

The VBScript **TextStream** object facilitates sequential access to a file.

Methods: VBScript TextStream Object

VBScript TextStream Object Close Method	Closes an open stream.
VBScript TextStream Object Read Method	Reads a specified number of characters from a stream.
VBScript TextStream Object ReadAll Method	Reads an entire stream.
VBScript TextStream Object ReadLine Method	Reads an entire line from a stream.
VBScript TextStream Object Skip Method	Skips a specified number of characters when reading a stream.
VBScript TextStream Object SkipLine Method	Skips the next line when reading a stream.
VBScript TextStream Object Write Method	Writes a specified string to a stream.
VBScript TextStream Object WriteBlankLines Method	Writes a specified number of newline characters to a stream.
VBScript TextStream Object WriteLine Method	Writes a specified string and newline character to a stream.

Properties: VBScript TextStream Object

VBScript TextStream Object AtEndOfLine Property	**True** if the file pointer is before the end-of-line marker.
VBScript TextStream Object AtEndOfStream Property	**True** if the file pointer is at the end of the stream

VBScript TextStream Object Column Property	The column number of the current character in the stream.
VBScript TextStream Object Line Property	The current line number of the stream.

Syntax: VBScript TextStream Object

```
TextStream.{property | method}
```

The property and method argument can be any of the properties and methods associated with the **TextStream** object.

Note

In actual usage, **TextStream** is replaced by a variable placeholder representing the **TextStream** object returned from the **FileSystemObject**.

Remarks: VBScript TextStream Object

In the following code, *a* is the **TextStream** object returned by the **CreateTextFile** method on the **FileSystemObject**:

```
Set fs = CreateObject("Scripting.FileSystemObject")
Set a = fs.CreateTextFile("c:\testfile.txt", True)
a.WriteLine("This is a test.")
a.Close
```

VBScript TextStream Object WriteLine Method and VBScript TextStream Object Close Method are two methods of the **TextStream** object.

VBScript TextStream Object AtEndOfLine Property

Read-only property that returns **True** if the file pointer immediately precedes the end-of-line marker in a **TextStream** file; **False** if it is not.

Syntax: VBScript TextStream Object AtEndOfLine Property

```
object.AtEndOfLine
```

Arguments: VBScript TextStream Object AtEndOfLine Property

object

The name of a **TextStream** object.

Remarks: VBScript TextStream Object AtEndOfLine Property

The **AtEndOfLine** property applies only to **TextStream** files that are open for reading; otherwise, an error occurs.

The following code illustrates the use of the **AtEndOfLine** property:

```
Dim fs, a, retstring
```

```
Set fs = CreateObject("Scripting.FileSystemObject")
Set a = fs.OpenTextFile("c:\testfile.txt", ForReading, False)
Do While a.AtEndOfLine <> True
retstring = a.Read(1)
...
Loop
a.Close
```

VBScript TextStream Object AtEndOfStream Property

Read-only property that returns **True** if the file pointer is at the end of a **TextStream** file; **False** if it is not.

Syntax: VBScript TextStream Object AtEndOfStream Property

```
object.AtEndOfStream
```

Arguments: VBScript TextStream Object AtEndOfStream Property

object

The name of a **TextStream** object.

Remarks: VBScript TextStream Object AtEndOfStream Property

The **AtEndOfStream** property applies only to **TextStream** files that are open for reading; otherwise, an error occurs.

The following code illustrates the use of the **AtEndOfStream** property:

```
Dim fs, a, retstring
Set fs = CreateObject("Scripting.FileSystemObject")
Set a = fs.OpenTextFile("c:\testfile.txt", ForReading, False)
Do While a.AtEndOfStream <> True
retstring = a.ReadLine
...
Loop
a.Close
```

VBScript TextStream Object Close Method

Closes an open **TextStream** file.

Syntax: VBScript TextStream Object Close Method

```
object.Close
```

Arguments: VBScript TextStream Object Close Method

object

The name of a **TextStream** object.

VBScript TextStream Object Column Property

Read-only property that returns the column number of the current character position in a **TextStream** file.

Syntax: VBScript TextStream Object Column Property

```
object.Column
```

Arguments: VBScript TextStream Object Column Property

object

The name of a **TextStream** object.

Remarks: VBScript TextStream Object Column Property

After a newline character has been written, but before any other character is written, **Column** is equal to 1.

VBScript TextStream Object Line Property

Read-only property that returns the current line number in a **TextStream** file.

Syntax: VBScript TextStream Object Line Property

```
object.Line
```

Arguments: VBScript TextStream Object Line Property

object

The name of a **TextStream** object.

Remarks: VBScript TextStream Object Line Property

After a file is initially opened and before anything is written, **Line** is equal to 1.

VBScript TextStream Object Read Method

Reads a specified number of characters from a **TextStream** file and returns the resulting string.

Syntax: VBScript TextStream Object Read Method

```
object.Read(characters)
```

Arguments: VBScript TextStream Object Read Method

object

The name of a **TextStream** object. Required.

characters

The number of characters you want to read from the file. Required.

VBScript TextStream Object ReadAll Method

Reads an entire **TextStream** file and returns the resulting string.

Syntax: VBScript TextStream Object ReadAll Method

```
object.ReadAll
```

Arguments: VBScript TextStream Object ReadAll Method

object

The name of a **TextStream** object.

Remarks: VBScript TextStream Object ReadAll Method

For large files, using the **ReadAll** method wastes memory resources. Other techniques should be used to input a file, such as reading a file line-by-line.

VBScript TextStream Object ReadLine Method

Reads an entire line (up to, but not including, the newline character) from a **TextStream** file and returns the resulting string.

Syntax: VBScript TextStream Object ReadLine Method

```
object.ReadLine
```

Arguments: VBScript TextStream Object ReadLine Method

object

The name of a **TextStream** object.

VBScript TextStream Object Skip Method

Skips a specified number of characters when reading a **TextStream** file.

Syntax: VBScript TextStream Object Skip Method

```
object.Skip(characters)
```

Arguments: VBScript TextStream Object Skip Method

object

The name of a **TextStream** object. Required.

characters

Number of characters to skip when reading a file. Required.

Remarks: VBScript TextStream Object Skip Method

Skipped characters are discarded.

VBScript TextStream Object SkipLine Method

Skips the next line when reading a **TextStream** file.

Syntax: VBScript TextStream Object SkipLine Method

```
object.SkipLine
```

Arguments: VBScript TextStream Object SkipLine Method

object

The name of a **TextStream** object.

Remarks: VBScript TextStream Object SkipLine Method

Skipping a line means reading and discarding all characters in a line up to and including the next newline character.

An error occurs if the file is not open for reading.

VBScript TextStream Object Write Method

Writes a specified string to a **TextStream** file.

Syntax: VBScript TextStream Object Write Method

```
object.Write(string)
```

Arguments: VBScript TextStream Object Write Method

object

The name of a **TextStream** object. Required.

string

The text you want to write to the file. Required.

Remarks: VBScript TextStream Object Write Method

Specified strings are written to the file with no intervening spaces or characters between each string. Use the VBScript TextStream Object WriteLine Method to write a newline character or a string that ends with a newline character.

VBScript TextStream Object WriteBlankLines Method

Writes a specified number of newline characters to a **TextStream** file.

Syntax: VBScript TextStream Object WriteBlankLines Method

```
object.WriteBlankLines(lines)
```

Arguments: VBScript TextStream Object WriteBlankLines Method

object

The name of a **TextStream** object.

lines

The number of newline characters you want to write to the file. Required.

Remarks: VBScript TextStream Object WriteBlankLines Method

For Windows systems, **WriteBlankLines** uses <CR><LF> as the newline character. On UNIX systems, **WriteBlankLines** uses <LF>.

VBScript TextStream Object WriteLine Method

Writes a specified string and newline character to a **TextStream** file.

Syntax: VBScript TextStream Object WriteLine Method

```
object.WriteLine([string])
```

Arguments: VBScript TextStream Object WriteLine Method

object

The name of a **TextStream** object. Required.

string

The text you want to write to the file. If omitted, a newline character is written to the file. Optional.

Remarks: VBScript TextStream Object WriteLine Method

For Windows systems, **WriteLine** uses <CR><LF> as the newline character. On UNIX systems, **WriteLine** uses <LF>.

VBScript FileSystemObject Collections

VBScript FileSystemObject Collections

VBScript Drives Collection	Read-only collection of available drives.
VBScript Files Collection	Collection of all **File** objects within a **Folder** object.
VBScript Folders Collection	Collection of all **Folder** objects within a **Folder** object.

Collection Methods

VBScript Add Method	Adds a new **Folder** object to a **Folders** collection.

Collection Properties

VBScript Count Property	The number of items in a collection.
VBScript Item Property	An item in a collection.

Note

Collections returned by **FileSystemObject** method calls reflect the state of the file system when the collection was created. Changes to the file system after creation are not reflected in the collection. If the file system might be changed during the lifetime of the collection object, the method returning the collection should be called again to ensure that the contents are current.

VBScript Drives Collection

Read-only collection of all available drives.

Methods: VBScript Drives Collection

None

Properties: VBScript Drives Collection

VBScript Count Property, VBScript Item Property

Remarks: VBScript Drives Collection

Removable-media drives need not have media inserted for them to appear in the **Drives** collection.

On UNIX systems the drives collection will contain only one member, the "/" drive.

The following code illustrates how to get the **Drives** collection and iterate the collection using the **For Each…Next** statement:

```
Sub ShowDriveList

Dim fs, d, dc, s, n

Set fs = CreateObject("Scripting.FileSystemObject")

Set dc = fs.Drives

For Each d in dc

s = s & d.DriveLetter & " - "

If d.DriveType = Remote Then

n = d.ShareName

Else

n = d.VolumeName
```

```
End If
s = s & n & vbCrLf
Next
Response.Write s
End Sub
```

VBScript Files Collection

Collection of all **File** objects within a folder.

Methods: VBScript Files Collection

None

Properties: VBScript Files Collection

VBScript Count Property, VBScript Item Property

Remarks: VBScript Files Collection

The following code illustrates how to get a **Files** collection and iterate the collection using the **For Each…Next** statement:

```
Sub ShowFolderList(folderspec)
Dim fs, f, f1, fc, s
Set fs = CreateObject("Scripting.FileSystemObject")
Set f = fs.GetFolder(folderspec)
Set fc = f.Files
For Each f1 in fc
s = s & f1.name
s = s & vbCrLf
Next
Response.Write s
End Sub
```

VBScript Folders Collection

Collection of all **Folder** objects contained within a **Folder** object.

Methods: VBScript Folders Collection

VBScript Add Method

Properties: VBScript Folders Collection

VBScript Count Property, VBScript Item Property

Remarks: VBScript Folders Collection

The following code illustrates how to get a **Folders** collection and how to iterate the collection using the **For Each…Next** statement:

```
Sub ShowFolderList(folderspec)
Dim fs, f, f1, fc, s
Set fs = CreateObject("Scripting.FileSystemObject")
Set f = fs.GetFolder(folderspec)
Set fc = f.SubFolders
For Each f1 in fc
s = s & f1.name
s = s & vbCrLf
Next
Response.Write s
End Sub
```

VBScript Add Method

Adds a new **Folder** to a **Folders** collection.

VBScript Add Method Applies To:

VBScript Folders Collection

Syntax: VBScript Folders Add Method

```
object.Add folderName
```

Arguments: VBScript Folders Add Method

object

Always the name of a **Folders** collection. Required.

folderName

The name of the new **Folder** being added. Required.

Remarks: VBScript Folders Add Method

An error occurs if the *folderName* already exists.

VBScript Count Property

Returns the number of items in a collection. Read-only.

VBScript Count Property Applies To:

VBScript Folders Collection

Syntax: VBScript Count Property

```
object.Count
```

Arguments: VBScript Count Property

object

The name of a collection.

Remarks: VBScript Count Property

The following code illustrates use of the **Count** property:

```
Dim a, d, i 'Create some variables
Set d = CreateObject("Scripting.Dictionary")
d.Add "a", "Athens" 'Add some keys and items.
d.Add "b", "Belgrade"
d.Add "c", "Cairo"
a = d.Keys 'Get the keys
For i = 0 To d.Count -1 'Iterate the array
Print a(i) 'Print key
Next
...
```

VBScript Item Property

Returns an item based on the specified key. Read/write.

Syntax: VBScript Item Property

```
object.Item(key) [= newitem]
```

Arguments: VBScript Item Property

object

The name of a collection. Required.

key

A *key* associated with the *item* being retrieved or added. Required.

Remarks: VBScript Item Property

If *key* is not found when attempting to return an existing *item,* a new *key* is created and its corresponding *item* is left empty.

Appendix A: Errors Reference

This appendix explains the error messages you might encounter when using Sun Chili!Soft ASP.

In this appendix:

Sun Chili!Soft ASP Errors

The following table describes Sun Chili!Soft ASP error messages:

Error Code	Error Name	Explanation
100	Out of memory	Unable to allocate required memory.
101	Unexpected error	The function returned "|."
102	Expecting string input	The function expects a string as input.
103	Expecting numeric input	The function expects a number as input.
104	Operation not allowed	The operation is not allowed.
105	Index out of range	An array index is out of range.
106	Type mismatch	An unhandled data type was encountered.
107	Stack overflow	The data being processed is over the allowed limit.
108	Create object failed	An error occurred while creating object "|."
109	Member not found	The member was not found.
110	Unknown name	The name is unknown.
111	Unknown interface	The interface is unknown.
112	Missing parameter	A parameter is missing.
113	Script timed out	The maximum amount of time for a script to execute was exceeded. You can change this limit by specifying a new value for the property **Server.ScriptTimeOut** or by changing the value in the ASP administration tools.
114	Object not free threaded	The application object accepts only free-threaded

		objects; object "|" is not free threaded.
115	Unexpected error	A trappable error occurred in an external object. The script cannot continue running.
116	Missing close of script delimiter	The script block lacks the close of script tag (%>).
117	Missing close of script tag	The script block lacks the close of script tag (</SCRIPT>) or close of tag symbol (>).
118	Missing close of object tag	The object block lacks the close of object tag (</OBJECT>) or close of tag symbol (>).
119	Missing Classid or Progid attribute	The object instance "|" requires a valid Classid or Progid in the object tag.
120	Invalid Runat attribute	The **Runat** attribute of the script tag or object tag can only have the value "Server."
121	Invalid scope in object tag	The object instance "|" cannot have application or session scope. To create the object instance with session or application scope, place the object tag in the Global.asa file.
122	Invalid scope in object tag	The object instance "|" must have application or session scope. This applies to all objects created in a Global.asa file.
123	Missing Id attribute	The required **Id** attribute of the object tag is missing.
124	Missing Language attribute	The required **Language** attribute of the script tag is missing.
125	Missing close of attribute	The value of the "|" attribute has no closing delimiter.
126	Include file not found	The Include file "|" was not found.
127	Missing close of HTML comment	The HTML comment or server-side include lacks the close tag (-->).
128	Missing File or Virtual attribute	The Include file name must be specified using either the **Missing File** or **Virtual** attribute.
129	Unknown scripting language	The scripting language "|" is not found on the server.
130	Invalid File attribute	**File** attribute "|" cannot start with forward slash or backslash.
131	Disallowed parent path	The Include file "|" cannot contain ".." to indicate the parent directory.
132	Compilation error	The Active Server Page "|" could not be processed.
133	Invalid ClassID attribute	The object tag has an invalid ClassID of "|."

134	Invalid ProgID attribute	The object has an invalid ProgID of "|."
135	Cyclic include	The file "|" is included by itself (perhaps indirectly). Please check Include files for other Include statements.
136	Invalid object instance name	The object instance "|" is attempting to use a reserved name. This name is used by Active Server Pages intrinsic objects.
137	Invalid global script	Script blocks must be one of the allowed Global.asa procedures. Script directives within <% ... %> are not allowed within the Global.asa file. The allowed procedure names are **Application_OnStart**, **Application_OnEnd**, **Session_OnStart**, or **Session_OnEnd**.
138	Nested script block	A script block cannot be placed inside another script block.
139	Nested object	An object tag cannot be placed inside another object tag.
140	Page command out Of order	The @ command must be the first command within the Active Server Page.
141	Page command repeated	The @ command can only be used once within the Active Server Page.
142	Thread token error	A thread token failed to open.
143	Invalid application name	A valid application name was not found.
144	Initialization error	The page level objects list failed during initialization.
145	New application failed	The new application could not be added.
146	New session failed	The new session could not be added.
147	500 server error	Server error 500.
148	Server too busy	Server too busy to service the request.
149	Application restarting	The request cannot be processed while the application is being restarted.
150	Application directory error	The application directory could not be opened.
151	Change notification error	The change notification event could not be created.
152	Security error	An error occurred while processing a user's security credentials.
153	Thread error	A new thread request failed.
154	Write HTTP header error	The HTTP headers could not be written to the client browser.

155	Write page content error	The page content could not be written to the client browser.
156	Header error	The HTTP headers are already written to the client browser. Any HTTP header modifications must be made before writing page content.
157	Buffering on	Buffering cannot be turned off once it is already turned on.
158	Missing URL	A URL is required.
159	Buffering off	Buffering must be on.
160	Logging failure	Failure to write entry to log.
161	Data type error	The conversion of a variant to a string variable failed.
162	Cannot modify cookie	The cookie "ASPSessionID" cannot be modified. It is a reserved cookie name.
163	Invalid comma use	Commas cannot be used within a log entry. Please select another delimiter.
164	Invalid TimeOut value	An invalid **TimeOut** value was specified.
165	SessionID error	A **SessionID** string cannot be created.
166	Uninitialized object	An attempt was made to access an uninitialized object.
167	Session initialization error	An error occurred while initializing the **Session** object.
168	Disallowed object use	An intrinsic object cannot be stored within the **Session** object.
169	Missing object information	An object with missing information cannot be stored in the **Session** object. The threading model information for an object is required.
170	Delete session error	The session did not delete properly.
171	Missing path	The **Path** parameter must be specified for the **MapPath** method.
172	Invalid path	The **Path** parameter for the **MapPath** method must be a virtual path. A physical path was used.
173	Invalid path character	An invalid character was specified in the Path parameter for the **MapPath** method.
174	Invalid path character(s)	An invalid "/" or "\\" was found in the **Path** parameter for the **MapPath** method.
175	Disallowed path characters	The ".." characters are not allowed in the **Path** parameter for the **MapPath** method.
176	Path not found	The **Path** parameter for the **MapPath** method did not

		correspond to a known path.
177	Server.CreateObject failed	The call to **Server.CreateObject** failed.
178	Server.CreateObject access error	The call to **Server.CreateObject** failed while checking permissions. Access is denied to this object.
179	Application initialization error	An error occurred while initializing the **Application** object.
180	Disallowed object use	An intrinsic object cannot be stored within the **Application** object.
181	Invalid threading model	An object using the apartment-threading model cannot be stored within the **Application** object.
182	Missing object information	An object with missing information cannot be stored in the **Application** object. The threading model information for the object is required.
183	Empty cookie key	A cookie with an empty key cannot be stored.
184	Missing cookie name	A name must be specified for a cookie.
185	Missing default property	A default property was not found for the object.
186	Error parsing certificate	There was an error parsing the certificate.
187	Object addition conflict	Could not add object to application. Application was locked down by another request for adding an object.
188	Disallowed object use	Cannot add objects created using object tags to the session intrinsic.
189	Disallowed object use	Cannot add objects created using object tags to the application intrinsic.
190	Unexpected error	A trappable error occurred while releasing an external object.
191	Unexpected error	A trappable error occurred in the **OnStartPage** method of an external object.
192	Unexpected error	A trappable error occurred in the **OnEndPage** method of an external object.
193	OnStartPage failed	An error occurred in the **OnStartPage** method of an external object.
194	OnEndPage failed	An error occurred in the **OnEndPage** method of an external object.
195	Invalid Server method call	This method of the **Server** object cannot be called during **Session_OnEnd** and **Application_OnEnd**.
196	Cannot launch out of process component	Only InProc server components should be used. If you want to use LocalServer components, you must set the

		AllowOutOfProcCmpnts registry setting. See the README file for important considerations.
197	Disallowed object use	Cannot add object with apartment model behavior to the application intrinsic object.
199	Disallowed object use	Cannot add JScript objects to the session.
200	Out of range "Expires" attribute	The date given for "Expires" precedes January 1, 1980, or exceeds Jan 19, 2038, 3:14:07 GMT.
201	Unknown scripting language in registry	The scripting language "|" specified in the registry is not found on the server.
202	Missing code page	The code page attribute is missing.
203	Invalid code page	The specified code page attribute is invalid.
204	Invalid CodePage value	An invalid **CodePage** value was specified.
205	Change notification	Failed to create event for change notification.
206	Cannot call BinaryRead	Cannot call **BinaryRead** after using **Request.Form** collection.
207	Cannot use Request.Form	Cannot use **Request.Form** collection after calling **BinaryRead**.
208	Cannot use generic Request collection	Cannot use the generic **Request** collection after calling **BinaryRead**.
210	Method not implemented	This method has not yet been implemented.
212	Cannot clear buffer	**Response.Clear** is not allowed after a **Response.Flush** while client debugging is enabled.
214	Invalid Path parameter	The **Path** parameter exceeds the maximum length allowed.
215	Illegal value for SESSION property	The **SESSION** property can only be TRUE or FALSE.
217	Invalid scope in object tag	Object scope must be Page, Session, or Application.
218	Missing LCID	The **LCID** attribute is missing.
219	Invalid LCID	The specified LCID is not available.
221	Invalid @ command directive	The specified "|" option is unknown or invalid.
222	Invalid TypeLib specification	METADATA tag contains an invalid Type Library specification.
223	TypeLib not found	METADATA tag contains a Type Library specification that does not match any registry entry.
224	Cannot load TypeLib	Cannot load Type Library specified in the

		METADATA tag.
225	Cannot wrap TypeLibs	Cannot create a **Type Library Wrapper** object from the Type Libraries specified in METADATA tags.
226	Cannot modify StaticObjects	Illegal assignment. **StaticObjects** collection cannot be modified at run time.
299	Unexpected error	The ASP engine has not been correctly registered.

VBScript Errors

The following table describes VBScript error messages:

Error Code	Message
5	Invalid procedure call or argument
6	Overflow
7	Out of memory
9	Subscript out of range
10	Array fixed or temporarily locked
11	Division by zero
13	Type mismatch
14	Out of string space
28	Out of stack space
35	Sub or Function not defined
48	Error in loading DLL
51	Internal error
53	File not found
57	Device I/O error
58	File already exists
61	Disk full
67	Too many files
70	Permission denied
75	Path/File access error
76	Path not found
91	Object variable or With block variable not set

92	For loop not initialized
94	Invalid use of **Null**
322	Cannot create necessary temporary file
424	Object required
429	ActiveX component cannot create object
430	Class does not support automation
432	File name or class name not found during automation operation
438	Object does not support this property or method
440	Automation error
445	Object does not support this action
446	Object does not support named arguments
447	Object does not support current locale setting
448	Named argument not found
449	Argument not optional
450	Wrong number of arguments or invalid property assignment
451	Object not a collection
453	Specified DLL function not found
455	Code resource lock error
457	This key already associated with an element of this collection
458	Variable uses an automation type not supported in VBScript
500	Variable is undefined
501	Illegal assignment
502	Object not safe for scripting
503	Object not safe for initializing
1001	Out of memory
1002	Syntax error
1003	Expected ":"
1004	Expected ";"
1005	Expected "("
1006	Expected ")"
1007	Expected "]"
1008	Expected "{"

1009	Expected "}"
1010	Expected identifier
1011	Expected "="
1012	Expected "If"
1013	Expected "To"
1014	Expected "End"
1015	Expected "Function"
1016	Expected "Sub"
1017	Expected "Then"
1018	Expected "Wend"
1019	Expected "Loop"
1020	Expected "Next"
1021	Expected "Case"
1022	Expected "Select"
1023	Expected expression
1024	Expected statement
1025	Expected end of statement
1026	Expected integer constant
1027	Expected "While" or "Until"
1028	Expected "While," "Until," or end of statement
1029	Too many locals or arguments
1030	Identifier too long
1031	Invalid number
1032	Invalid character
1033	Unterminated string constant
1034	Unterminated comment
1035	Nested comment
1037	Invalid use of "Me" keyword
1038	"Loop" without "Do"
1039	Invalid "Exit" statement
1040	Invalid "For" loop control variable
1041	Name redefined

1042	Must be first statement on the line
1043	Cannot assign to non-**ByVal** argument
1044	Cannot use parentheses when calling a Sub
1045	Expected literal constant
1046	Expected "In"
32766	True
32767	False
32811	Element not found

JScript Errors

The following table describes JScript error messages:

Error Code	Message
5	Invalid procedure call or argument
6	Overflow
7	Out of memory
9	Subscript out of range
10	This array is fixed or temporarily locked
11	Division by zero
13	Type mismatch
14	Out of string space
17	Cannot perform requested operation
28	Out of stack space
35	Sub or Function not defined
48	Error in loading DLL
51	Internal error
52	Bad file name or number
53	File not found
54	Bad file mode
55	File already open
57	Device I/O error
58	File already exists

61	Disk full
62	Input past end of file
67	Too many files
68	Device unavailable
70	Permission denied
71	Disk not ready
74	Cannot rename with different drive
75	Path/File access error
76	Path not found
91	Object variable or With block variable not set
92	For loop not initialized
93	Invalid pattern string
94	Invalid use of **Null**
322	Cannot create necessary temporary file
424	Object required
429	Automation server cannot create object
430	Class does not support automation
432	File name or class name not found during automation operation
438	Object doesn't support property or method <item>
440	Automation error
445	Object doesn't support this action
446	Object doesn't support named arguments
447	Object doesn't support current locale setting
448	Named argument not found
449	Argument not optional
450	Wrong number of arguments or invalid property assignment
451	Object not a collection
453	Specified DLL function not found
458	Variable uses an automation type not supported in JScript
501	Cannot assign to variable
502	Object not safe for scripting
503	Object not safe for initializing

504	Object not safe for creating
5000	Cannot assign to "this"
5001	<Item> is not a number; number expected
5002	<Item> is not a function; function expected
5003	Cannot assign to a function result
5004	<Item> is not an object that can be indexed; cannot index object
5005	<Item> is not a string; string expected
5006	<Item> is not a **Date** object; **Date** object expected
5007	<Item> is not an object; object expected
5008	Cannot assign to <item>; illegal assignment
5009	<Item> is undefined; undefined identifier
5010	<Item> is not a Boolean; Boolean expected
5011	Cannot execute code from a freed script
5012	Cannot delete <item>; object member expected
5013	<Item> is not a VBArray; VBArray expected
5014	<Item> is not a JScript object; JScript object expected
5015	<Item> is not an **Enumerator** object; **Enumerator** object expected
5016	<Item> is not a **RegularExpression** object; **RegularExpression** object expected
5017	Syntax error in regular expression
5018	Unexpected quantifier
5019	Expected "]" in regular expression
5020	Expected ")" in regular expression
5021	Invalid range in character set

ADO Errors

The following table describes ADO error messages:

Constant Name	Number	Description
adErrInvalidArgument	3001	The application is using arguments that are of the wrong type, are out of acceptable range, or are in conflict with one another.

adErrNoCurrentRecord	3021	Either BOF or EOF is TRUE, or the current record has been deleted; the operation request by the application requires a current record.
adErrIllegalOperation	3219	The operation requested by the application is not allowed in this context.
adErrFeatureNotAvailable	3251	The operation requested by the application is not supported by the provider.
adErrItemNotFound	3265	ADO could not find the object in the collection corresponding to the name or ordinal reference requested by the application.
adErrObjectInCollection	3367	Cannot append. Object already in collection.
adErrObjectNotSet	3420	The object referenced by the application no longer points to a valid object.
adErrDataConversion	3421	The application is using a value of the wrong type for the current operation.
adErrObjectClosed	3704	The operation requested by the application is not allowed if the object is closed.
adErrObjectOpen	3705	The operation requested by the application is not allowed if the object is open.
adErrProviderNotFound	3706	ADO could not find the specific provider.
adErrBoundToCommand	3707	The application cannot change the **ActiveConnection** property of a **Recordset** object with a **Command** object as its source.
adErrInvalidParamInfo	3708	The application has improperly defined a **Parameter** object.
adErrInvalidConnection	3709	The application requested an operation on an object with a reference to a closed or invalid **Connection** object.

Appendix B: Troubleshooting

This appendix provides troubleshooting tips for problems you might encounter when running Sun Chili!Soft ASP.

In this appendix:

ASP Pages not Served

Issue: My browser asks if I want to download or open ASP pages instead of serving them.

Resolution: Either your Sun Chili!Soft ASP installation did not complete properly, or you made changes to your Web server configuration files that removed the Sun Chili!Soft ASP configuration settings. If you have made no changes to your Web server configuration files, reinstalling Sun Chili!Soft ASP should correct this problem.

If you modified your Web server configuration settings, and you are running iPlanet Web Server, then you might not have selected the option to **Save** and **Apply** manual changes in the iPlanet Administrator. If so, the Sun Chili!Soft ASP modifications to the configuration files have been removed. In this case, you can either reinstall Sun Chili!Soft ASP or reinstate the Sun Chili!Soft ASP configuration settings, which are described in "Configuring the Web Server After Installation" in "Chapter 2: Installing and Configuring Sun Chili!Soft ASP."

Can't Connect to a Database

Issue: I cannot connect to my Oracle or Sybase database.

Resolution: Make sure that the **ServerName** parameter for your Data Source Name (DSN) is set to point to the server that is running the client software for your database and not to the database server itself. For more information about configuring DSNs, see "Configuring Data Source Names (DSNs)," in "Chapter 3: Managing Sun Chili!Soft ASP."

Issue: I cannot connect to a DB2 database.

Resolution: If your DB2 database server is configured to run in a different locale than your ASP Server and Web server, you might get the following error message when trying to connect to it from an ASP page:

"There is no available conversion for the source code page "932" to the target code page "1252". Reason Code "1". SQLSTATE=57017"

To address this problem, create the DB2 database on a server configured for the same locale in which the Web server and the Sun Chili!Soft ASP Server are configured to run. For more information, see "Configuring International Support" in "Chapter 3: Managing Sun Chili!Soft ASP."

Can't Compile Apache

Issue: I cannot compile Apache Web Server with Sun Chili!Soft ASP.

Resolution: Not all versions of Apache Web Server are supported by Sun Chili!Soft ASP. Make sure that you are using a version of Apache that is supported by your version of Sun Chili!Soft ASP, as listed in "Supported Platforms and Web Servers" in "Chapter 1: About Sun Chili!Soft ASP."

FileSystemObject Permission Denied Error

Issue: I get a "Permission Denied" error when I use the **FileSystemObject**.

Resolution: First, make sure that the user account that Sun Chili!Soft ASP is running under has permission to open the file. Next, check the setting of the **EnableParentPaths** registry setting. If **EnableParentPaths** is set to **False**, your scripts are only allowed to access files in or beneath the directory that defines the ASP application for which you are calling the **FileSystemObject**.

Appendix C: Glossary

A:

Absolute path name

Active Server component

Active Server Pages (ASP)

Active Template Library (ATL)

ActiveX

ActiveX controls

ActiveX Data Objects (ADO)

ActiveX scripting

Administration Console

ADO

Adobe GoLive

Apache Web Server

Application object

Application server

Application Service Provider (ASP)

Argument

ASP application

ASP component

ASP engine

ASP page

ASP script

ASP Server

ASP session

ASP technology

Built-in objects

C++

CAB

Cabinet (CAB)

Cache

CASP

CGI

Chili!Beans

CIFS

Client-side script

Cobalt

Code page

COM

COM+

Common Gateway Interface (CGI)

Common Internet File System (CIFS)

Common Object Request Broker Architecture (CORBA)

Component

Component Object Model (COM)

Connection pooling

Connection string

Cookie

CORBA

Database server

Data connection

Data Source Name (DSN)

dBASE

Dedicated hosting

Design-time control

Document Object Model (DOM)

DSN

DSN-less connection string

DSO

Dynamic Link Library (DLL)

Dynamic Shared Object (DSO)

ECMAScript (European Computer Manufacturers' Association Script)

Enterprise JavaBeans (EJB)

Event

Expression

Extensible Markup Language (XML)

Extensible Stylesheet Language (XSL)

File DSN

File upload

FileSystem object

FrontPage

FrontPage Server Extensions

Function

Global.asa

Hostname

HP-UX

HTML

HTTP server

IBM AIX

IBM DB2

Interbase

Internationalization

Internet Services Application Programming Interface (ISAPI)

iPlanet Web Server, Enterprise Edition

ISAPI

Java

Java applet

JavaBeans

Java Database Connectivity (JDBC)

JavaScript

Java Virtual Machine (JVM)

JDBC

JScript

Key

Linux

Localization

Local Language Identifier (LCID)

Macromedia UltraDev

Method

Microsoft Access

Microsoft Data Access Components

Microsoft FrontPage

Microsoft Internet Information Server (IIS)

Microsoft JScript

Microsoft SQL Server

MTS

Microsoft Transaction Server (MTS)

Microsoft Visual Basic

Microsoft Visual Basic Scripting Edition (VBScript)

Microsoft Visual InterDev

Moniker

Multi-threading

MyODBC

Zeus Web Server

Absolute path name

In a computer operating system, a path is the route through a file system to a particular file. A path name (or pathname in Windows) is the specification of that path, including the name of the file. An absolute path name (or fully qualified path name) specifies the complete path name. A relative path name specifies a path relative to the directory to which the operating system is currently set.

Each operating system has its own format for specifying a path name. The DOS, Windows, and OS/2 operating systems use this format:

Drive_letter:\directoryname\subdirectoryname\filename.suffix

In UNIX-based systems, the format is:

/directory/subdirectory/filename

In UNIX, the storage drive location is not an explicit part of the path name.

Active Server component

An Active Server component runs on the server side as part of an ASP application. Active Server components are activated through Active Server Pages (ASP) technology, but do not require a Windows interface.

Active Server Pages (ASP)

Active Server Pages (ASP) is a specification for a dynamically created Web page having an .asp extension. ASP technology provides an open, compile-free application environment in which Web developers can combine HTML, scripts, and reusable Active Server components. A Sun Chili!Soft ASP page uses VBScript or JScript code to access the ASP object model, which exposes functionality that is often used in Web application environments. When a browser requests an ASP page, the Web server passes execution to the Sun Chili!Soft ASP Server, which processes the scripts, generates an HTML page, and sends it back to the browser.

See also:

What is ASP? in "Chapter 1: About Sun Chili!Soft ASP"

What is Sun Chili!Soft ASP? in "Chapter 1: About Sun Chili!Soft ASP"

Active Template Library (ATL)

The Active Template Library (ATL) is a set of template-based C++ classes. With ATL, you can create objects and specify the threading model: single-threaded, apartment, free-threaded, or both free-threaded and apartment. ATL simplifies the programming of Component Object Model (COM) objects by providing special support for key COM features, such as **IUnknown**, **IClassFactory**, **IClassFactory2**, and **Idispatch**, dual interfaces, standard COM enumerator interfaces, connection points, tear-off interfaces, and ActiveX controls.

ActiveX

ActiveX is a set of technologies built on the Component Object Model (COM) that enable software components, regardless of the language in which they were developed, to work together in a networked environment. Although ActiveX technologies are mainly used to develop interactive Web content, they can also be used in desktop applications and other programs.

ActiveX controls

ActiveX controls are reusable, stand-alone software components that often expose a subset of the total functionality of a product or application. They were formerly referred to as OLE controls or OCX. ActiveX controls cannot run stand-alone. They must be loaded into a control container, such as Visual Basic or Internet Explorer. An ActiveX control can also be embedded in a Visual C++ resource.

ActiveX Data Objects (ADO)

ActiveX Data Objects (ADO) was designed by Microsoft to be a high-level interface for data objects. ADO can be used to access many different types of data, including Web pages, spreadsheets, and other types of documents. Within ASP, ADO is most commonly used in conjunction with ODBC drivers to connect to databases and other data sources available through ODBC drivers. Sun Chili!Soft ASP includes its own implementation of ADO, which supports all of the commonly used functionality found in Microsoft ADO 2.0 and some of the popular functionality found in Microsoft ADO 2.5.

See also:

ADO Component Reference in "Chapter 5: Developer’s Reference"

ActiveX scripting

ActiveX scripting controls the integrated behavior of ActiveX controls and/or Java applets from the server or the browser. To enable server-side scripting, such as with ASP, ActiveX scripting requires that the appropriate interpreter for the scripting language be installed on the server. Sun Chili!Soft ASP includes script interpreters for both Visual Basic Scripting Edition (VBScript) and JScript.

ADO

ADO, short for ActiveX Data Objects, is a high-level interface that Microsoft developed for data objects. ADO can be used to access many different types of data, including Web pages, spreadsheets, and other types of documents. Within ASP, ADO is most commonly used in conjunction with ODBC drivers to connect to databases and other data sources available through ODBC drivers. Sun Chili!Soft ASP includes its own implementation of ADO, which supports all of the commonly used functionality found in Microsoft ADO 2.0 and some of the popular functionality found in Microsoft ADO 2.5.

See also:

Connecting to a Database in "Chapter 4: Building a Sun Chili!Soft ASP Application"

ADO Component Reference in "Chapter 5: Developer’s Reference"

Adobe GoLive

Adobe GoLive is a Web development tool for ASP applications that uses a WYSIWYG ("What You See Is What You Get") interface. This is one of several Web development tools appropriate for the beginning to mid-level ASP developer.

See also:

Macromedia UltraDev

Microsoft FrontPage

Apache Web Server

Apache Web Server (Apache HTTP Server) is a widely used Web server that is developed and maintained through an open-source project. It is freely available for all major UNIX platforms and for Windows. Sun Chili!Soft ASP supports Apache Web Server.

Application object

The **Application** object is one of the five built-in objects included in Sun Chili!Soft ASP. The **Application** object stores variables and objects that are available to scripts running within the scope of an ASP application. A simple example of how you can use this object would be to store a counter that tracks the number of users (and thus sessions) currently active for a given ASP application.

See also:

Using Sun Chili!Soft ASP Built-in Objects in "Chapter 4: Building a Sun Chili!Soft ASP Application"

Application server

An application server is a program that handles all Web application operations between Web browsers and back-end business or database servers. Because many databases cannot interpret commands written in HTML, the application server works as a translator by retrieving data from databases and using business logic encapsulated in code to output dynamically generated HTML code. The Sun Chili!Soft ASP Server is an application server that enables you to use the Active Server Pages (ASP) specification to execute database queries, execute business logic, and generate the presentation layer for Web applications.

Application Service Provider (ASP)

Application Service Providers (ASPs) manage and distribute software-based services and solutions to customers across a wide area network from a central data center. This is not to be confused with Active Server Pages, which shares the same acronym (ASP).

Argument

In object-oriented programming an argument is a value passed from one function to another. Methods can take one or more arguments, or none at all. Arguments can be optional, in which

case you do not need to enter anything for an argument. If an argument is optional, all arguments following it are also optional.

Administration Console

The Sun Chili!Soft ASP Administration Console is a browser-based tool that enables Sun Chili!Soft ASP administrators to configure and control the Sun Chili!Soft ASP Server and its bindings to Web servers and database servers. The Administration Console consists of an ASP application that uses its own Web server and Sun Chili!Soft ASP Server to administer the primary ASP Server.

See also:

Using the Administration Console in "Chapter 3: Managing Sun Chili!Soft ASP"

ASP application

An ASP application is a set of Active Server Pages (ASP) files contained within a single root directory. For the Sun Chili!Soft ASP Server to recognize an ASP application, the root directory must be defined as an application on the ASP Server or as a virtual directory on the Web server. The files within the root directory of an ASP application share the same global.asa file, which must be contained in the root directory itself, rather than in a subdirectory under the root. All variables and objects for an ASP application are scoped from the root directory.

See also:

Creating the Basic ASP Application in "Chapter 4: Building a Sun Chili!Soft ASP Application"

ASP component

An ASP component is designed to run on a Web server as part of an ASP application. ASP components provide key functionality needed for Web applications, such as database access, so that developers do not need to create and re-create the code to perform these tasks. ASP components do not require browser-scripting ability, so they are useful for implementing tasks that are difficult to accomplish with browser scripting.

See also:

Using Sun Chili!Soft ASP Built-in Objects in "Chapter 4: Building a Sun Chili!Soft ASP Application"

Using Sun Chili!Soft ASP Installed Components in "Chapter 4: Building a Sun Chili!Soft ASP Application"

ASP engine

The Sun Chili!Soft ASP Server uses an ASP engine to execute ASP scripts. A single ASP engine executes as many threads as are specified in the Sun Chili!Soft ASP Administration Console.

See also:

Configuring Multi-threading in "Chapter 3: Managing Sun Chili!Soft ASP"

ASP page

The first step in building an ASP application is creating an ASP page. An ASP page is simply a plain text file with the .asp filename extension.

An ASP page contains optional text (usually HTML and/or client-side scripts) interspersed with one or more script blocks. To create an ASP page, you insert script commands into an HTML page. With Sun Chili!Soft ASP, you can write scripts in Visual Basic Scripting Edition (VBScript) or JScript. Any valid HTML page can be a valid ASP page, enabling the Web developer to easily transform a static Web site into a dynamic Web site by adding ASP scripts to existing documents. Saving the page with an .asp filename extension tells the Web server how to process the script commands.

See also:

What is ASP? in "Chapter 1: About Sun Chili!Soft ASP"

What is Sun Chili!Soft ASP? in "Chapter 1: About Sun Chili!Soft ASP"

ASP script

An ASP script is a server-side script that is included on an Active Server Pages (ASP) page. With Sun Chili!Soft ASP, scripts can be written in either Visual Basic Scripting Edition (VBScript) or JScript.

See also:

What is ASP? in "Chapter 1: About Sun Chili!Soft ASP"

What is Sun Chili!Soft ASP? in "Chapter 1: About Sun Chili!Soft ASP"

ASP Server

When a browser requests an ASP page, the Web server passes execution to the Sun Chili!Soft ASP Server, which processes the HTML code and ASP scripts on the page, generates an HTML page, and sends the page back to the browser.

See also:

Managing the ASP Server in "Chapter 3: Managing Sun Chili!Soft ASP."

ASP session

An ASP session is created by using the Sun Chili!Soft ASP built-in Session object, which uses ASP technology to share information about a user between Web pages. As the user navigates between the pages of a site, information about the user is maintained through a cookie.

See also:

Managing User Sessions in "Chapter 4: Building a Sun Chili!Soft ASP Application"

ASP technology

ASP, short for Active Server Pages, is a specification for a dynamically created Web page having an .asp extension. ASP technology provides an open, compile-free application environment in

which Web developers can combine HTML, scripts, and reusable Active Server components. A Sun Chili!Soft ASP page uses scripts written in Visual Basic Scripting Edition (VBScript) or JScript to access the ASP object model, which exposes functionality that is often used in Web application environments. When a browser requests an ASP page, the Web server passes execution to the ASP Server, which uses an ASP script to generate an HTML page. The ASP Server then sends the page back to the browser.

See also:

What is ASP? in "Chapter 1: About Sun Chili!Soft ASP"

What is Sun Chili!Soft ASP? in "Chapter 1: About Sun Chili!Soft ASP"

Built-in Objects

Sun Chili!Soft ASP includes five built-in objects--**Application**, **Request**, **Response**, **Server**, and **Session**--that handle many common programming tasks. These objects enable you to avoid much of the overhead associated with complex Web programming, so you can focus on creating interesting, interactive Web content rather than on doing low-level programming.

See also:

Using Sun Chili!Soft ASP Built-in Objects in "Chapter 4: Building a Sun Chili!Soft ASP Application"

ASP Built-in Objects Reference in "Chapter 5: Developer's Reference"

C++

C++ is a high-level, object-oriented version of the C programming language. C++ is one of the most popular programming languages for graphical applications that run on systems that have graphical user interfaces.

CAB

CAB, short for cabinet, is a technology for compression and distribution of files. When used for Java applets, the CAB file serves as a single, compressed repository for all .class files and all audio and image data required by the applet. Only the CAB file is downloaded, so the time of download is the time it takes to negotiate the transfer and download the compressed bytes. Once downloaded, the contents of the CAB file are extracted and installed.

Cabinet (CAB)

Cabinet (CAB) is a technology for compression and distribution of files. When used for Java applets, the CAB file serves as a single, compressed repository for all .class files and all audio and image data required by the applet. Only the CAB file is downloaded, so the time of download is the time it takes to negotiate the transfer and download the compressed bytes. Once downloaded, the contents of the CAB file are extracted and installed.

Cache

A cache is a high-speed storage mechanism. It can either be a reserved section of main memory or an independent high-speed storage device. Sun Chili!Soft ASP supports memory caching of ASP scripts in a tokenized format that dramatically increases performance.

See also:

Enabling Script Caching in "Chapter 3: Managing Sun Chili!Soft ASP"

CASP

CASP is a commonly used abbreviation for Sun Chili!Soft ASP. It is used for the default installation directory name and for several virtual directories installed by the Sun Chili!Soft ASP setup program.

CGI

CGI, short for Common Gateway Interface, is a server-side interface for initiating software services. Software that handles input and output in accordance with the CGI standard is considered a CGI application. For example, when a user submits a form through a Web browser, the server executes an application, known as a CGI script, and passes the user's input information to that application by using CGI. The application then returns information to the server by using CGI. Active Server Pages (ASP) technology is a high-performance alternative to CGI.

Chili!Beans

Sun Chili!Beans enables access to Java classes (including JavaBeans and Enterprise JavaBeans) from the ASP environment by wrapping Java classes in a Component Object Model (COM) layer. This enables Java objects to be used by COM controllers, such as ActiveX scripting engines like VBScript. Chili!Beans is included in Sun Chili!Soft ASP. Chili!Beans is designed to work with any Java Virtual Machine (JVM) that implements the Java Native Interface (JNI) specification, such as JDK 1.2.x or 1.3.x.

See also:

Chili!Beans Component Reference in "Chapter 5: Developer's Reference"

Sun Chili!Soft ASP Administration Console

The Sun Chili!Soft ASP Administration Console is a browser-based tool that enables Sun Chili!Soft ASP administrators to configure and control the Sun Chili!Soft ASP Server and its bindings to Web servers and database servers. The Administration Console consists of an ASP application that uses its own Web server and Sun Chili!Soft ASP Server to administer the primary ASP Server.

See also:

Using the Administration Console in "Chapter 3: Managing Sun Chili!Soft ASP"

CIFS

CIFS, short for Common Internet File System, is an open, cross-platform technology that defines a standard remote file system access protocol for use over the Internet. CIFS enables groups of users to work together and share documents across the Internet or within their corporate intranets regardless of their computer or operating system platform. CIFS runs over TCP/IP and uses the Internet's global Domain Naming Service (DNS) for scalability, and is specifically optimized to support slower speed dial-up connections common on the Internet.

Client-side script

A client-side script is part of an HTML document that is downloaded to the user's browser. The browser interprets and executes the script on the client computer. Benefits of client-side scripting include increased speed, and the ability to make a page act more like a real application by connecting embedded components and responding to events. Client-side scripting can also be used in applications that are not Web-based, but which use Dynamic HTML. Examples of client-side scripting languages include VBScript, JavaScript, JScript, and ECMAScript. Unlike server-side scripting, client-side scripting requires no special implementation on the Web server. However, it does require the appropriate support on the client browser. Sun Chili!Soft ASP applications involve server-side scripts and can also include client-side scripts.

See also:

Adding Scripts in "Chapter 4: Building a Sun Chili!Soft ASP Application"

Cobalt

Sun Cobalt is a leading developer of server appliances. The Sun Cobalt product line includes Sun Cobalt RaQ XTR, Sun Cobalt Qube, Sun Cobalt Cache, Sun Cobalt RaQ, and Chili!Soft ASP. These products are used by businesses, service providers, and educational institutions for Internet and Web hosting services. Founded in 1996, Cobalt Networks was aquired by Sun Microsystems in December of 2000.

Code page

A code page is a character set that a computer uses to interpret and display data properly. Code pages usually correspond to different platforms and languages and are important in international applications. Your system administrator can set the correct code page and LCID for a given language by using the Sun Chili!Soft ASP Administration Console.

See also:

Developing International Applications in "Chapter 4: Building a Sun Chili!Soft ASP Application"

COM

COM, short for Component Object Model, is a model for binary code developed by Microsoft that enables programmers to develop objects that can be accessed by any COM-compliant application. Both OLE and ActiveX are based on COM. In COM, client software accesses an object through a pointer to an interface, or a related set of functions called methods, on the object. COM components can be written in a variety of programming languages. One advantage of COM

is that it gives ASP scripts the ability to dynamically bind to COM objects, thereby linking simple scripting (VBScript or JScript) to compiled objects (typically written in C or C++). On Linux and UNIX, Sun Chili!Soft ASP uses its own proprietary COM emulation layer.

COM+

COM+ is an extension of COM that makes it easier for developers to create and use software components in any language, using any tool, by building on COM's integrated services and features. Delivered on the Microsoft Windows platform, COM+ extends developers' current investments in COM, offering new optional services, such as database access.

Common Gateway Interface (CGI)

Common Gateway Interface (CGI) is a server-side interface for initiating software services. Software that handles input and output in accordance with the CGI standard is considered a CGI application. For example, when a user submits a form through a Web browser, the server executes an application, known as a CGI script, and passes the user's input information to that application by using CGI. The application then returns information to the server by using CGI. Active Server Pages (ASP) technology is a high-performance alternative to CGI.

Common Internet File System (CIFS)

Common Internet File System (CIFS) is an open, cross-platform technology that defines a standard remote file system access protocol for use over the Internet. CIFS enables groups of users to work together and share documents across the Internet or within their corporate intranets regardless of their computer or operating system platform. CIFS runs over TCP/IP and utilizes the Internet's global Domain Naming Service (DNS) for scalability, and is specifically optimized to support slower speed dial-up connections common on the Internet.

Common Object Request Broker Architecture (CORBA)

Common Object Request Broker Architecture (CORBA) enables pieces of applications, called objects, to communicate with one another regardless of which programming language they were written in or on which operating system they're running. An industry consortium known as the Object Management Group (OMG) developed CORBA. There are several implementations of CORBA, the most widely used being IBM's SOM and DSOM architectures. Two competing models are Microsoft COM and the Sun Microsystems RMI. CORBA objects can be accessed by using Chili!Beans. For more information about using CORBA objects within Sun Chili!Soft ASP, contact Sun Chili!Soft Customer Support.

Component

A component is an object that encapsulates both data and code, and provides a well-specified set of publicly available services. Components encapsulate the business logic in your ASP applications. You can create your own components or buy them "off the shelf." Once you have a component, you can reuse it wherever it’s needed. You can develop components using C++ or Java.

Component Object Model (COM)

The Component Object Model (COM) is a model for binary code developed by Microsoft. COM enables programmers to develop objects that can be accessed by any COM-compliant application. Both OLE and ActiveX are based on COM. In COM, client software accesses an object through a pointer to an interface, or a related set of functions, called methods, on the object. COM components can be written in a variety of programming languages. One advantage of COM is that it gives ASP scripts the ability to dynamically bind to COM objects, thereby linking simple scripting (VBScript or JScript) to compiled objects (typically written in C or C++). On UNIX and Linux, Sun Chili!Soft ASP uses its own proprietary COM emulation layer.

Connection pooling

To improve server performance, you can configure the Sun Chili!Soft ASP Server to share open database connections among multiple users who are accessing the Web application. This is called database connection pooling. With connection pooling, rather than opening and closing a database connection for each individual request, the ASP Server uses a connection that is already open.

See also:

Setting the ADO Connection Pool Size in "Chapter 3: Managing Sun Chili!Soft ASP"

Connection string

A connection string defines the source of data for an external database. On a Sun Chili!Soft ASP page, a connection string must either include values for all required parameters for the database, or one of the following:

- A reference to a system DSN that the system administrator has defined for the database
- A reference to a file DSN that defines the required parameters for the database

See also:

Creating Connection Strings in "Chapter 4: Building a Sun Chili!Soft ASP Application"

Cookie

A cookie is a file of encoded information, stored on a user's computer, which identifies the user's computer during the current and subsequent visits to a Web site.

CORBA

CORBA, short for Common Object Request Broker Architecture, enables pieces of applications, called objects, to communicate with one another regardless of what programming language they were written in or on which operating system they're running. An industry consortium known as the Object Management Group (OMG) developed CORBA. There are several implementations of CORBA, the most widely used being IBM's SOM and DSOM architectures. Two competing models are Microsoft COM and the Sun Microsystems RMI. CORBA objects can be accessed by

using Chili!Beans. For more information about using CORBA objects within Sun Chili!Soft ASP, contact Sun Chili!Soft Customer Support.

Database server

A database server exclusively serves database connections.

Data connection

A data connection is a collection of information required to access a specific database. This information includes a data source name (DSN) and logon information. For example, a data connection for a Microsoft SQL Server database consists of the name of the database, the location of the server on which it resides, network information used to access that server, a user ID, and a password.

See also:

Creating Database Connections on the Server in "Chapter 2: Installing and Configuring Sun Chili!Soft ASP"

Configuring a Database in "Chapter 3: Managing Sun Chili!Soft ASP"

Connecting to a Database in "Chapter 4: Building a Sun Chili!Soft ASP Application"

Data Source Name (DSN)

A Data Source Name (DSN) refers to a collection of information required to connect an ASP application to a particular ODBC-compliant database. The ODBC Driver Manager uses this information to create the database connection. Sun Chili!Soft ASP supports two types of DSNs: system DSNs and file DSNs.

See also:

Creating Database Connections on the Server in "Chapter 2: Installing and Configuring Sun Chili!Soft ASP"

Configuring a Database in "Chapter 3: Managing Sun Chili!Soft ASP"

Connecting to a Database in "Chapter 4: Building a Sun Chili!Soft ASP Application"

dBASE

dBASE is a database management system. Sun Chili!Soft ASP includes an ODBC driver for connecting to dBASE 5.

Dedicated hosting

Dedicated hosting refers to the practice of dedicating the resources of an entire server or group of servers to a specific Web site. This helps maintain the performance of high-traffic, high-volume Web sites because server resources are not shared with other Web sites.

Design-time control

Design-time controls are visual design components written in ActiveX that help developers construct dynamic Web applications by automatically generating standard HTML and/or scripting code at design time, instead of at run time. Design-time controls found in the many development tools that Sun Chili!Soft ASP supports generate code that is compatible with Sun Chili!Soft ASP. These tools include Adobe GoLive, Microsoft FrontPage and Visual Interdev, Macromedia UltraDev, and others.

Document Object Model (DOM)

The Document Object Model (DOM) is a programming interface specification of the World Wide Web Consortium (W3C) that enables a programmer to create and modify HTML pages and XML documents as full-fledged program objects. Currently, HTML (Hypertext Markup Language) and XML (Extensible Markup Language) can be used to express a document in terms of a data structure. By defining documents as program objects, their contents and data can be "hidden" within the object, helping to ensure control over who can manipulate the document. As objects, documents can carry with them the object-oriented procedures called methods. DOM is a strategic and open effort to specify how to provide programming control over documents. It was inspired in part by the advent of the new HTML capabilities generally called dynamic HTML, and as a way to encourage consistent browser behavior with Web pages and their elements.

DSN

A Data Source Name (DSN) refers to a collection of information required to connect an ASP application to a particular ODBC-compliant database. The ODBC Driver Manager uses this information to create the database connection. Sun Chili!Soft ASP supports two types of DSNs: system DSNs and file DSNs.

See also:

Creating Database Connections on the Server in "Chapter 2: Installing and Configuring Sun Chili!Soft ASP"

Configuring a Database in "Chapter 3: Managing Sun Chili!Soft ASP"

Connecting to a Database in "Chapter 4: Building a Sun Chili!Soft ASP Application"

DSN-less connection string

A DSN-less connection string is a connection string that includes all of the information needed for connecting to a data source, rather than incorporating the information by reference to a system DSN or a file DSN. DSN-less connection strings enable you to move the ASP application from one server to another without recreating a system DSN on the new server. File DSNs provide this advantage as well. Note that when migrating ASP applications from one environment to another, such as from Windows to UNIX or Linux, you must make changes to your connection strings so that they work in the new environment.

See also:

Enabling Database Connections on the Server in "Chapter 2: Installing and Configuring Sun Chili!Soft ASP"

Configuring a Database in "Chapter 3: Managing Sun Chili!Soft ASP"

Connecting to a Database in "Chapter 4: Building a Sun Chili!Soft ASP Application"

DSO

Sun Chili!Soft ASP communicates with Apache Web Server through an object module. In the DSO (Dynamic Shared Object) version, Apache Web Server object module entries are loaded on an as-needed basis.

Dynamic link library (DLL)

A dynamic link library (DLL) is a code file containing functions that can be called from other executable code (either an application or another DLL). Programmers use DLLs so they can reuse code and parcel out distinct jobs. Unlike an executable (EXE) file, a DLL cannot be directly run. DLLs must be called from other code that is already executing. DLLs and EXEs apply only to Windows environments. In UNIX environments, the Sun Chili!Soft ASP engine and all components run on a Windows emulation layer (either Mainsoft MainWin or the Chili!Soft proprietary emulation layer). The equivalent of a DLL on these emulation layers are files having an *.so (Shared Object) filename extension.

Dynamic Shared Object (DSO)

Sun Chili!Soft ASP communicates with Apache Web Server through an object module. In the Dynamic Shared Object (DSO) version, Apache Web Server object module entries are loaded on an as-needed basis.

ECMAScript

ECMAScript, short for European Computer Manufacturers Association Script, is a scripting language based on JavaScript that meets the ECMA-262 standard. ECMA, like other scripting languages, enriches and enlivens Web pages, but is the only scripting language on the Web based on a standard. The ECMA-262 specification outlines an object-oriented programming language that performs computations and manipulates objects within a host environment, such as the browser.

Enterprise JavaBeans (EJB)

Enterprise JavaBeans (EJB) is an architecture for setting up program components written in the Java programming language that run in the server parts of a computer network running under the client/server model. Sun Chili!Soft ASP enables ASP applications to access EJB through Chili!Beans.

EJB is built on the JavaBeans technology for distributing program components (called beans, using the coffee metaphor) to clients in a network. EJB offers enterprises the advantage of being able to control changes at the server, instead of needing to update each individual client whenever a new program component is changed or added. EJB components are reusable in multiple applications. To deploy an EJB or component, it must be part of a specific application, called a container.

Originated by Sun Microsystems, EJB is roughly equivalent to the Microsoft Component Object Model/Distributed Component Object Model architecture. However, like all Java-based

architectures, EJB-based applications can be deployed across all major operating systems, and not just Windows. EJB program components are generally known as servlets (little server programs). The application or container that runs the servlets is sometimes called an application server. A typical use of servlets is to replace Web programs that use the Common Gateway Interface (CGI) and a Perl script. Another general use is providing an interface between Web users and a legacy mainframe application and its database.

With EJB, there are two types of beans: session beans and entity beans. An entity bean is described as one that, unlike a session bean, has persistence and can retain its original behavior or state.

See also:

Chili!Beans Component Reference in "Chapter 5: Developer's Reference"

Event

In application programming, an event is a notification that occurs in response to some action. It can be a change in state; the result of the user clicking or moving the mouse or pressing a keyboard key; or other actions that are focus-related, element-specific, or object-specific. Programmers write code that responds to these actions. In Web development, HTML events are triggered and handled by code that is executed in the browser, and thus don't generally apply to ASP. Strictly speaking, only events that require a round-trip back to the Web server involve ASP code.

Expression

In application programming, expression is any combination of operators, constants, literal values, functions, names of columns, controls, and properties that result in a single value.

Extensible Markup Language (XML)

Extensible Markup Language (XML) is a simplified subset of Standard Generalized Markup Language (SGML) that provides a file format for representing data, a method for describing data structure, and a mechanism for extending and annotating HTML with semantic information. Allowing an unlimited set of tags, XML tags indicate what kind of data each tag contains, rather than indicating how something should look. For instance, a tag might hold a price, an order number, or a name. The flexibility of XML allows the document's author to determine what kind of data to use, and to choose the tag types that most fit the author's needs.

As a universal data format, XML provides a standard for the server-to-server transfer of different types of structured data so that the information can be decoded, manipulated, and displayed consistently and correctly. In addition, it enables the development of three-tier Web applications, acting as the data transfer format between the middle-tier Web server and the client.

Extensible Stylesheet Language (XSL)

Extensible Stylesheet Language (XSL) is a style-sheet language that defines the rules for mapping structured XML data and documents. Derived from Document Style Semantics and Specification Language (DSSSL), XSL also has roots in the Standard Generalized Markup Language (SGML) community.

Using XSL, an element can be formatted and displayed in multiple places on a Web page, or rearranged or removed from the page. Developers can then generate a presentation structure that may be quite different from the original data structure. XSL does not replace Cascading Style Sheets (CSS); rather, it is designed to handle the new capabilities of XML that CSS cannot. Although CSS can be used to display simple XML data, CSS is not general enough to handle all of the possibilities generated by XML; the syntax of XSL can.

File DSN

A file DSN (Data Source Name) refers to a collection of information, in the form of parameters and their values, required for connecting an ASP application to a particular database. The information is contained within a file having a *.dsn filename extension. Developers can incorporate the database information into a connection string by referring to the file DSN rather than needing to specify the values for each required parameter. A file DSN can be shared among several users. Sun Chili!Soft ASP also supports system DSNs.

See also:

Enabling Database Connections on the Server in "Chapter 2: Installing and Configuring Sun Chili!Soft ASP"

Connecting to a Database in "Chapter 4: Building a Sun Chili!Soft ASP Application"

FileSystem object

In ASP applications, the **FileSystem** object provides access to a computer's file system. The **FileSystem** object can perform simple functions such as opening and closing files. Various methods can be applied to the **FileSystem** object to affect the organization and properties of a file system.

See also:

JScript FileSystemObject Object in "JScript Language Reference" in "Chapter 5: Developer’s Reference"

VBScript FileSystemObject Object in "VBScript Language Reference" in "Chapter 5: Developer’s Reference"

File upload

File upload refers to the transmission of a file from one computer system to another. To upload is to send a file to another computer, and to download is to receive a file.

FrontPage

FrontPage is a Web site creation and management tool provided by Microsoft. FrontPage utilizes WYSIWYG (What You See Is What You Get) editing and graphical interfaces, and includes wizards, templates, and drag-and-drop functionality.

Note

Sun Chili!Soft ASP enables you to run ASP pages generated by Microsoft FrontPage. Specific questions about the installation, configuration, and use of FrontPage and FrontPage Server Extensions should be directed to Microsoft or its representatives.

See also:

Enabling FrontPage Publishing in "Chapter 3: Managing Sun Chili!Soft ASP"

Using FrontPage Database Features in "Chapter 4: Building a Sun Chili!Soft ASP Application"

FrontPage Server Extensions

Sun Chili!Soft ASP supports but no longer installs Microsoft FrontPage Server Extensions. With FrontPage Server Extensions, Web authors and developers working on Windows-based computers can use the FrontPage client to publish Web pages and applications to UNIX- or Linux-based Web servers, or to Windows NT- and Windows 2000-based computers running a Web server other than Internet Information Server (IIS).

Note

Sun Chili!Soft ASP enables you to run ASP pages generated by Microsoft FrontPage. Specific questions about the installation, configuration, and use of FrontPage and FrontPage Server Extensions should be directed to Microsoft or its representatives.

See also:

Enabling FrontPage Publishing in "Chapter 3: Managing Sun Chili!Soft ASP"

Function

In application programming, a function is the general term used for a bit of code that performs a specific, limited task. Functions (also known as subroutines) enable the programmer to divide complex tasks into smaller, more manageable pieces. Problems can be isolated more readily since each subroutine or function can be tested separately.

Global.asa

Global.asa is a file that contains information that is global to a specific ASP application. This file is read and its application and session variables are initialized before the first ASP page in a given ASP application is read. Global.asa enables the developer to specify event procedures and declare objects that have session or application scope.

See also:

Using the global.asa File in "Chapter 4: Building a Sun Chili!Soft ASP Application"

Hostname

A hostname is the valid DNS name for a Web server.

On Apache Web Server 1.3 and newer, the hostname is defined by the **ServerName** directive, which you can find in the Apache http.conf file. The syntax is as follows:

```
ServerName [WEB_SERVER_HOSTNAME]
```

On iPlanet/Netscape Enterprise 3.6, you can find the hostname on the Netscape Administration Web site under **Server Preferences** --> **View Server Settings**. Alternatively, you can find the hostname in the iPlanet/Netscape magnus.conf file under `ServerName`. The syntax is as follows:

```
ServerName [WEB_SERVER_HOSTNAME]
```

HP-UX

HP-UX is an acronym for Hewlett-Packard UNIX, a version of UNIX developed by Hewlett-Packard for use on the company's internal workstations and now available as a commercial product.

HTML

HTML (Hypertext Markup Language) is the set of markup symbols or codes inserted in a file intended for display on a Web page. The markup tells the Web browser how to display text and images. Each individual markup code is referred to as an element or tag. Some elements come in pairs, which indicate when some display effect is to begin and when it is to end.

HTML is a formal Recommendation by the World Wide Web Consortium (W3C) and is generally adhered to by the major browsers, Netscape Navigator and Microsoft Internet Explorer. The current version of HTML is HTML 4.0. However, both Internet Explorer and Netscape implement some features differently and provide nonstandard extensions. Web developers using the more advanced features of HTML 4.0 might need to design pages for both browsers and send out the appropriate version to each user. Dynamic HTML (DHTML) is a significant feature in HTML 4.0. What is sometimes referred to as HTML 5.0 is an extensible form of HTML called Extensible Hypertext Markup Language (XHTML).

HTTP server

An HTTP server, also called a Web server, uses the Hypertext Transfer Protocol (HTTP) to provide information in hypertext format. Client software relays this input from the user to the server and displays information from the server in HTTP format. Other types of Internet-based servers include File Transfer Protocol (FTP) and Gopher. The Web is a network consisting of these types of servers. The most popular HTTP servers are Apache Web Server, iPlanet Web Server Enterprise Edition (formerly Netscape), Zeus Web Server, and Microsoft Internet Information Server (IIS).

IBM AIX

IBM AIX is an open operating system from IBM that is based on a version of UNIX. AIX/ESA was designed for IBM System/390 or large server hardware platform. AIX/6000 is an operating system that runs on the IBM workstation platform, the RISC System/6000.

IBM DB2

IBM DB2 is a relational database management system for large business computers. DB2 products are offered for UNIX-based systems and personal-computer operating systems. DB2 databases can be accessed from any application program by using ActiveX Data Objects (ADO) and the Microsoft Open Database Connectivity (ODBC) interface, the Java Database Connectivity (JDBC) interface, or a Common Object Request Broker Architecture (CORBA) interface broker.

Interbase

Interbase is an open-source relational database. Originally developed in 1985 by Borland Corporation, Interbase is offered with free development and distribution rights.

Internationalization

Internationalization, also called localization, is the process of making your Web site or application available in multiple language formats across the world.

See also:

Configuring International Support in "Chapter 3: Managing Sun Chili!Soft ASP"

Developing International Applications in "Chapter 4: Building a Sun Chili!Soft ASP Application"

Internet Services Application Programming Interface (ISAPI)

Internet Services Application Programming Interface (ISAPI) is a specification for extending Web server functionality in the Windows environment. An ISAPI extension is a DLL that exports specific functions according to the ISAPI specification. There are two types of ISAPI extensions: ISAPI filters and ISAPI applications. ISAPI provides comparable functionality to the Common Gateway Interface (CGI), but offers performance improvements on Windows-based Web servers.

iPlanet Web Server, Enterprise Edition

iPlanet Web Server, Enterprise Edition is Web server software originally developed by Netscape and now offered by iPlanet.

ISAPI

ISAPI, short for Internet Services Application Programming Interface, is a specification for extending Web server functionality in the Windows environment. An ISAPI extension is a DLL that exports specific functions according to the ISAPI specification. There are two types of ISAPI extensions: ISAPI filters and ISAPI applications. ISAPI provides comparable functionality to the Common Gateway Interface (CGI), but offers performance improvements on Windows-based Web servers.

Java

Developed by Sun Microsystems, Java is an object-oriented programming language, similar to C++. Java-based applications, or applets, can be quickly downloaded from a Web site and run using a Java-compatible Web browser such as Netscape Navigator or Microsoft Internet Explorer.

Java programs, or source code files (.java), are compiled into a format known as bytecode files (.class). Once compiled, these files can be executed by a Java interpreter. Most operating systems, including Windows, Macintosh OS, and UNIX have Java interpreters and run-time environments known as Java Virtual Machines.

See also:

Using Java Objects and Classes in "Chapter 4: Building a Sun Chili!Soft ASP Application"

Java applet

A Java applet is an HTML-based program built with Java, which a browser temporarily downloads to and runs from a user's hard disk. Java applets can be downloaded and run by any Java-interpreting Web browser, such as Netscape Navigator and Microsoft Internet Explorer. Java applets can be used to add multimedia effects (such as background music, real-time video displays, and animation), and interactivity (such as calculators and games) to Web pages.

See also:

Using Java Objects and Classes in "Chapter 4: Building a Sun Chili!Soft ASP Application"

JavaBeans

JavaBeans is an object-oriented programming interface developed by Sun Microsystems that enables developers to build reuseable applications or program building blocks called components, which can be deployed on any major operating system platform. Like Java applets, JavaBeans components (called beans) give Web pages (or other applications) interactive capabilities such as computing interest rates or varying page content based on user or browser characteristics. Sun Chili!Soft ASP enables ASP applications to access Java objects and classes through Chili!Beans.

From a user's point-of-view, a component can be a button that you interact with or a small calculating program that is initiated when you press the button. From a developer's point-of-view, the button component and the calculator component are created separately and can then be used together or in different combinations with other components in different applications or situations.

When the components or beans are in use, the properties of a bean (for example, the background color of a window) are visible to other beans. Beans that haven't "met" before can learn each other's properties dynamically, and interact accordingly.

Beans are developed by using a Beans Development Kit (BDK) from Sun Microsystems. They can be run on any major operating system platform inside a number of application environments (known as containers), including browsers, word processors, and other applications.

To build a component with JavaBeans, you write language statements using the Java programming language and include JavaBeans statements that describe component properties,

such as user interface characteristics and events that trigger a bean to communicate with other beans in the same container or elsewhere in the network.

Beans also have persistence, which is a mechanism for storing the state of a component in a safe place. This would allow, for example, a component (bean) to "remember" data that a particular user had already entered in an earlier user session.

JavaBeans gives Java applications the compound document capability that the OpenDoc and ActiveX interfaces already provide.

See also:

Using Java Objects and Classes in "Chapter 4: Building a Sun Chili!Soft ASP Application"

Java Database Connectivity (JDBC)

Java Database Connectivity (JDBC) is a data access interface based on OBDC and used with the Java programming language.

JavaScript

JavaScript is a scripting language that interacts with HTML source code and is compatible with the Java programming language. JavaScript is the Netscape implementation of the ECMA-262 standard.

Java Virtual Machine (JVM)

The Java Virtual Machine is the cornerstone of the Sun Microsystem Java programming language. It is the component of the Java technology responsible for Java's cross-platform delivery, the small size of its compiled code, and its ability to protect users from malicious programs.

The Java Virtual Machine is an abstract computing machine. Like a real computing machine, it has an instruction set and uses various memory areas. It is reasonably common to implement a programming language using a virtual machine; the best-known virtual machine may be the P-Code machine of UCSD Pascal. The Java Virtual Machine does not assume any particular implementation technology or host platform. It is not inherently interpreted, and it may just as well be implemented by compiling its instruction set to that of a real CPU, as for a conventional programming language. It may also be implemented in microcode, or directly in silicon.

The Java Virtual Machine knows nothing of the Java programming language, only of a particular file format, the .class file format. A .class file contains Java Virtual Machine instructions (or bytecodes) and a symbol table, as well as other ancillary information.

JDBC

JDBC, short for Java Database Connectivity, is a data-access interface based on OBDC and used with the Java programming language.

JScript

The Microsoft version of JavaScript, JScript is a standard scripting language based on the ECMA-262 standard. JScript is specifically targeted for the Internet and is built into Internet Explorer browsers. JScript is implemented as a fast, portable, lightweight interpreter that processes source code embedded directly in the HTML. JScript code does not produce stand-alone applets, but it is used to add interactivity to HTML documents. JScript uses syntax and language features similar to the Java, C, and C++ programming languages.

See also:

JScript Language Reference in "Chapter 5: Developer's Reference"

Key

In application programming, a key is the code for deciphering encrypted data.

Linux

Linux is an open source, UNIX-like operating system that runs on a variety of hardware platforms and is distributed free or at low cost. Unlike proprietary operating systems, Linux is publicly open and extendible by contributors. Linux's kernel (the central part of the operating system) was developed by Linus Torvalds. Linux is also distributed commercially by a number of companies.

Localization

Localization is the process of adapting a Web site or application so it is appropriate for the geographic area, or locale, in which it is used. If you are providing a Web site that will be viewed in countries other than the United States, you can set the default locale for the Web server by using the Sun Chili!Soft ASP Administration Console. Within an ASP page, you can use the CODEPAGE tag within the script delimiters to specify the proper code page.

See also:

Configuring International Support in "Chapter 3: Managing Sun Chili!Soft ASP"

Developing International Applications in "Chapter 4: Building a Sun Chili!Soft ASP Application"

Local Language Identifier (LCID)

A Local Language Identifier (LCID) is a 32-bit value that identifies a geographic locale. An LCID consists of a LangID and a sort key ID. The system administrator can set the LCID in Sun Chili!Soft ASP by using the Administration Console.

See also:

Configuring International Support in "Chapter 3: Managing Sun Chili!Soft ASP"

Developing International Applications in "Chapter 4: Building a Sun Chili!Soft ASP Application"

Macromedia UltraDev

Macromedia UltraDev is a version of Macromedia Dreamweaver that enables you to develop Web sites and Web applications that, among other things, connect to a database; incorporate ASP,

JSP and ColdFusion applications; and customize the Web site to display personalized content. The UltraDev comes with many more advanced development tools not offered in the regular version of Macromedia Dreamweaver.

Method

In application programming, a method is a logical operation provided by an object. In object-oriented programming, an object invokes a method by sending a message that contains the receiving object and the name of the specific method to invoke. Often, the name of the method is called a selector. Objects use messages as the mechanism through which they interact.

Microsoft Access

Microsoft Access is an ODBC-compliant database application included in the Microsoft Office Suite.

Microsoft Data Access Components

Microsoft Data Access Components are comprised of ADO and Remote Data Service (RDS), Open Database Connectivity (ODBC), and the Microsoft OLE DB Provider for ODBC, which are released, documented, and supported together. Of these, Sun Chili!Soft ASP supports ADO and ODBC only.

Microsoft FrontPage

Microsoft FrontPage is a Web site creation and management tool. FrontPage uses WYSIWYG (What You See Is What You Get) editing and graphical interfaces, and includes wizards, templates, and drag-and-drop functionality.

Microsoft Internet Information Server or Internet Information Services (IIS)

Microsoft Internet Information Server or Internet Information Services (IIS) is an HTTP server that runs on Windows NT Server 4.0 and Windows 2000 Server. An important feature of IIS is its Active Server Pages (ASP) capability, which enables Web authors to combine HTML, server-side scripts, and ActiveX controls. Sun Chili!Soft ASP brings ASP functionality to many other platforms in addition to Windows NT and Windows 2000 with IIS.

Microsoft JScript

The Microsoft version of JavaScript, JScript is a scripting language based on the ECMA-262 standard. JScript is specifically targeted for the Internet and built into Internet Explorer browsers. The JScript interpreter processes source code embedded directly in the HTML. JScript code does not produce stand-alone applets, but it is used to add interactivity to HTML documents. JScript uses syntax and language features similar to the Java, C, and C++ programming languages.

See also:

JScript Language Reference in "Chapter 5: Developer’s Reference"

Microsoft SQL Server

Microsoft SQL Server is Structured Query Language (SQL) server software offered by Microsoft. Sun Chili!Soft ASP is fully compatible with Microsoft SQL Server systems.

Microsoft Transaction Server (MTS)

Microsoft Transaction Server (MTS) is a component-based transaction processing system available on the Windows platform only. MTS defines an application-programming model for developing distributed, components-based applications, and provides a run-time infrastructure for deploying and managing these applications. Sun Chili!Soft ASP does not support MTS.

Microsoft Visual Basic

Microsoft Visual Basic is a high-level, visual programming language based on the BASIC (Beginner's All-purpose Symbolic Instruction Code) language and designed for building Windows-based applications. Visual Basic was one of the first products to provide a graphical programming environment and a paint metaphor for developing user interfaces. By dragging and dropping controls, such as buttons and dialog boxes, and then defining their appearance and behavior, the Visual Basic programmer is able to add a substantial amount of code without getting bogged down in syntactical details.

Although Visual Basic is not considered a true object-oriented programming language, it does embrace an object-oriented philosophy. It is sometimes called an event-driven language, because each object can react to different events.

Microsoft Visual Basic Scripting Edition (VBScript)

Microsoft Visual Basic Scripting Edition (VBScript) is a scripting language based on the Visual Basic programming language. Similar to both JScript and JavaScript, VBScript enables Web authors to include interactive controls, such as buttons and scroll bars, on their Web pages. For a Visual Basic programmer, VBScript is the easiest scripting language to learn. (JScript is easier for C/C++ programmers.) Sun Chili!Soft ASP supports both VBScript and JScript.

See also:

VBScript Language Reference in "Chapter 5: Developer’s Reference"

Microsoft Visual InterDev

Microsoft Visual InterDev is a team-based tool designed for developing HTML-based, data-driven, cross-platform Web applications for the Internet and corporate intranets. Visual InterDev contains a WYSIWYG (What You See Is What You Get) HTML editor, support for Dynamic HTML, client- and server-side debugging, database tools, and support for team-based development. Visual InterDev is also fully interoperable with Microsoft FrontPage for effective workgroup development of Web applications.

Moniker

A moniker is a name that uniquely identifies a Component Object Model (COM) object. Monikers support an operation known as binding, which is the process of locating the object

named by the moniker, activating it or loading it in memory if it isn't already there, and returning an interface pointer to it.

MTS

MTS, short for Microsoft Transaction Server, is a component-based transaction processing system available on the Windows platform only. MTS defines an application-programming model for developing distributed, components-based applications, and provides a run-time infrastructure for deploying and managing these applications. Sun Chili!Soft ASP does not support MTS.

Multi-threading

Multi-threading refers to running several processes in rapid sequence within a single program, regardless of which logical method of multi-tasking is being used by the operating system. Because the user's sense of time is much slower than the processing speed of a computer, the impression of multi-tasking appears simultaneous, even though only one task at a time can use a computer processing cycle.

See also:

Threading

MyODBC

MyODBC is the ODBC driver for the MySQL database server produced by TCX Data-Consult in Sweden. MySQL provides support for ODBC by means of the MyODBC program.

MyODBC allows you to:

- Connect to a remote database server from wherever you have access to a desktop application.
- Export a database to the remote server, and import a database from the remote server.
- Link a local database to a remote database.

See also:

Configuring a Database in "Chapter 3: Managing Sun Chili!Soft ASP"

MySQL Parameters in "Chapter 3: Managing Sun Chili!Soft ASP"

MySQL

MySQL is an open source database and relational database management system. MySQL uses an implementation of Structured Query Language (SQL).

See also:

Configuring a Database in "Chapter 3: Managing Sun Chili!Soft ASP"

MySQL Parameters in "Chapter 3: Managing Sun Chili!Soft ASP"

Netscape Enterprise Web Server

See iPlanet Web Server, Enterprise Edition.

NSAPI

NSAPI, short for Netscape Server API, is the API for Netscape Web servers. NSAPI enables programmers to create Web-based applications that are more sophisticated and run much faster than applications based on CGI.

Object

In object-oriented programming, an object is a variable comprising both routines and data that is treated as a discrete entity. An object is based on a specific model, where a client using an object's services gains access to the object's data through an interface consisting of a set of methods or related functions. The client can then call these methods to perform desired operations.

See also:

Object-oriented programming

Object-oriented programming

In object-oriented programming, an application is viewed as a collection of discrete objects (self-contained collections of data structures and routines that interact with other objects).

Object model

In application programming, the object model is the set of rules that makes an object perform a specific task. The object model is the structural foundation for object-oriented programming languages, such as C++.

Open Database Connectivity (ODBC)

Open Database Connectivity (ODBC) is a standard protocol for database servers. ODBC provides a common language for ASP applications to gain access to databases on a network. UNIX and Linux versions of Sun Chili!Soft ASP include ODBC drivers for a number of different databases. ODBC drivers enable you to connect to the databases and access their data.

See also:

Configuring a Database in "Chapter 3: Managing Sun Chili!Soft ASP"

ODBC driver

An ODBC driver is a module that enables a database to be accessed through ODBC. Each type of database (MySQL, Informix, DB2, and so forth) requires its own ODBC driver. Sun Chili!Soft ASP enables you to specify ODBC drivers by using the Sun Chili!Soft ASP Administration Console.

See also:

Configuring a Database in "Chapter 3: Managing Sun Chili!Soft ASP"

ODBC Manager

The ODBC Manager manages connections between ODBC drivers and databases by using information stored in the Data Source Name (DSN).

See also:

Configuring a Database in "Chapter 3: Managing Sun Chili!Soft ASP"

Parameter

In programming, a parameter is a value given to a variable. A parameter acts as placeholder in a query or stored procedure that can be filled in when the query or stored procedure is executed. Parameters enable you to use the same query or stored procedure many times, each time with different values.

Path name

In a computer operating system, a path is the route through a file system to a particular file. A path name (or pathname in Windows) is the specification of that path, including the name of the file. An absolute path name (or fully qualified path name) specifies the complete path name. A relative path name specifies a path relative to the directory to which the operating system is currently set.

Each operating system has its own format for specifying a path name. The DOS, Windows, and OS/2 operating systems use this format:

Drive_letter:directoryname\subdirectoryname\filename.suffix

In UNIX-based systems, the format is:

/directory/subdirectory/filename

In UNIX, the storage drive location is not an explicit part of the path name.

POP3

POP3, short for Post Office Protocol, is a protocol used to retrieve e-mail from a mail server. Most e-mail applications (also called e-mail clients) use the POP protocol, although some can use the newer IMAP (Internet Message Access Protocol). POP3 is supported in the Sun Chili!Soft SpicePack with the Chili!Mail component.

Port (TCP/IP)

A TCP/IP port is a "logical connection place." Using the Internet protocol, TCP/IP, a port enables a client program to specify a particular server program on a computer in a network. Higher-level applications that use TCP/IP, such as Hypertext Transfer Protocol (HTTP), have port numbers that are preassigned by the Internet Assigned Numbers Authority (IANA). These port numbers are called "well-known ports." Other application processes are assigned port numbers dynamically for each connection. When a service (or server program) is started initially, it is said to "bind" to its designated port number. Any client program that wants to use that server must send a request to bind to the designated port number.

Port numbers are between 0 and 65536. Ports 0 to 1024 are reserved for use by certain privileged services. For the HTTP service, port 80 is the default and does not need to be specified in the Uniform Resource Locator (URL).

Portability

Portability is a characteristic attributed to a computer application if that application can run on an operating system other than the one for which it was developed, without requiring a major rework. Porting software to a different operating system involves doing any work required to make the computer run in the new environment, such as resolving programming language differences, converting data, and adapting to new system procedures for running an application.

In general, applications that use standard application programming interfaces (APIs) such as the X/Open UNIX 95 standard C language interface, are portable. Ideally, such applications can simply be compiled for the operating system to which they are being ported. However, if an application uses operating system extensions or special capabilities that are not present in the new operating system, these features must be replaced with comparable ones in the new operating system.

Porting software typically involves some work. However, the Java programming language and runtime environment makes it possible to develop applications that run on any operating system supporting the Java standard (from Sun Microsystems) without any porting work. Java applets in the form of precompiled bytecode can be sent from a server program in one operating system to a client program (such as a Web browser) running on another operating system without change. Sun Chili!Soft ASP supports Java classes through Chili!Beans.

See also:

Using Java Objects and Classes in "Chapter 4: Building a Sun Chili!Soft ASP Application"

PostgreSQL

PostgreSQL is a sophisticated Object-Relational Database Management System (DBMS), supporting almost all SQL constructs, including subselects, transactions, and user-defined types and functions.

Procedural-based programming

Procedural-based programming is a method of programming that predates object-oriented programming. In this method, a programmer writes instructions that are followed by a computer from start to finish. Procedural-based programming requires that all instructions within the program follow a set of procedures, which are organized in a strict order and follow logical rules of procedure.

Process

A process is an instance of an application running on a computer. On UNIX and some other operating systems, a process is started when a program is initiated (either by a user entering a shell command or by another program). An application that is being shared by multiple users generally has one process for each user at some stage of execution.

Property

In application programming, a property is a named attribute of an object. Properties define object characteristics, such as size and name, or the state of an object, such as enabled or disabled. Properties do not take any arguments. All properties return a value; however, some properties are read-only, and some are read/write.

Request object

The **Request** object is a Sun Chili!Soft ASP built-in object that retrieves the values the client browser passed to the server during an HTTP request.

See also:

ASP Request Object in "Chapter 5: Developer's Reference"

Response object

The **Response** object is a Sun Chili!Soft ASP built-in object that you can use to control output sent to the client.

See also:

ASP Response Object in "Chapter 5: Developer's Reference"

Root directory

A root directory is the point of entry into the directory tree in a disk-based hierarchical directory structure. Branching from the root are various directories and subdirectories, each of which can contain one or more files and subdirectories.

Scripting

Scripting is a set of instructions for performing a special task in an application or utility, or on a Web site. Scripting languages are an intermediate stage between HTML and programming languages such as Java, C++, and Visual Basic. The primary difference between scripting languages and programming languages is that the syntax and rules of scripting languages are less rigid and intricate than those of programming languages. On the Web, scripts are processed either by the client ("client-side scripting") or the server ("server-side scripting"). Examples of scripting languages are Perl, JavaScript, VBScript, and JScript. Sun Chili!Soft ASP supports VBScript and JScript.

See also:

Adding Scripts in "Chapter 4: Building a Sun Chili!Soft ASP Application"

Server-side script

A server-side script is interpreted and executed by the server, and the results are sent to the browser. ASP scripts are server-side scripts. With Sun Chili!Soft ASP, when a browser requests an ASP page, the Web server sends the request to the Sun Chili!Soft ASP Server. The Sun Chili!Soft ASP Server reads the HTML and interprets and executes the script code, and then sends the resulting page to the browser.

Unlike client-side scripting, server-side scripting enables you to deliver highly customized Web pages without requiring any scripting intelligence on the client side. In fact, server-side scripting enables users to receive customized pages based on browser capabilities, user preferences, and content from a server-side database.

See also:

Adding Scripts in "Chapter 4: Building a Sun Chili!Soft ASP Application"

Secure Sockets Layer (SSL)

Secure Sockets Layer (SSL) is a security standard developed by Netscape Communications to secure application protocols such as HTTP over the Internet. SSL uses a key exchange method (RSA is most common) to establish an environment in which all data exchanged is encrypted with a cipher and hashed to protect it from eavesdropping and alteration. Primarily used for handling commerce payments, SSL is the most widely deployed security protocol on the Internet today. The Internet Engineering Task Force (IETF) has generated a successor of SSL, a network standard called Transport Layer Security (TLS).

SequeLink

SequeLink is server-based middleware provided by DataDirect Technologies (formerly a business unit of MERANT). SequeLink delivers optimized performance and reduced administration through a single, universal ODBC client and the SequeLink Common Server DBMS interface, removing the need for gateways and DBMS vendor middleware. Sun Chili!Soft ASP includes the SequeLink client for connecting to remote Microsoft Access and Microsoft SQL Server 6.5 databases running on Windows systems.

See also:

Configuring SequeLink in "Chapter 3: Managing Sun Chili!Soft ASP"

Servlet

A servlet is a small application that runs on a server. The term was coined in the context of the Java applet, a small application that is sent as a separate file along with a Web (HTML) page. Java applets usually run on a client and provide services such as performing a calculation for a user or positioning an image based on user interaction.

Server Name

Server Name (or ServerName) is a parameter specifying the name given to a specific database server either on the Internet or within an intranet. Sometimes referred to as a "friendly name," this name is a string of letters that gives the server an identity and resolves to the IP address of the computer running the database server.

Server object

The **Server** object is a Sun Chili!Soft ASP built-in object that provides access to methods and properties on the server. Most of its methods and properties serve as utility functions.

See also:

ASP Server Object in "Chapter 5: Developer's Reference"

Session object

The **Session** object is a Sun Chili!Soft ASP built-in object that stores information needed for a particular user session. Variables stored in the **Session** object are not discarded when the user jumps between pages in the application; instead, they persist for the entire user session. The Sun Chili!Soft ASP Server automatically creates a **Session** object when a user who does not already have a session requests an ASP page. The server destroys the **Session** object when the session expires or is abandoned. The **Session** object enables the ASP developer to:

- Automatically identify and classify requests coming from a single browser client into a logical application "session" on the server.
- Store session-scoped data on the server for use across multiple browser requests.
- Use session lifetime management events (**OnSessionStart**, **OnSessionEnd**).
- Automatically release session information if the browser does not revisit an application after a specified timeout period.

See also:

ASP Session Object in "Chapter 5: Developer's Reference"

Session state

Session state refers to information that the Sun Chili!Soft ASP Server stores in the ASP **Session** object about a sequence of requests that all come from the same browser. HTTP is a stateless protocol, meaning that it provides no way for the ASP Server to keep track of session state. As a result, ASP applications that maintain session-state information (such as shopping carts and data scrolling) require this type of infrastructure help.

See also:

ASP Session Object in "Chapter 5: Developer's Reference"

Shared hosting

Shared hosting refers to a Web-hosting environment where multiple Web sites share the resources of a single server (or group of servers) by means of virtual hosts. These Web sites may have different domain names, but they all ultimately resolve to the same IP address on the computer hosting the Web sites.

See also:

Running Sun Chili!Soft ASP in a Shared Web Hosting Environment in "Chapter 3: Managing Sun Chili!Soft ASP"

SMTP

SMTP, short for Simple Mail Transfer Protocol, is a TCP/IP (Transmission Control Protocol/Internet Protocol) protocol used for sending and receiving e-mail. However, because SMTP is limited in its ability to queue messages at the receiving end, it is usually used with one of two other protocols: POP3 (Post Office Protocol 3) or IMAP (Internet Message Access Protocol). Those protocols enable the user to save messages in a server mailbox and download them periodically from the server. Messaging applications typically use SMTP for sending e-mail and either POP3 or IMAP for downloading messages that have been received for them by the mail server.

SMTP is usually implemented to operate over TCP (Transmission Control Protocol) port 25. You can find the details of SMTP in Request for Comments 821 of the IETF (Internet Engineering Task Force). An alternative to SMTP, widely used in Europe, is X.400.

See also:

Chili!Mail (SMTP) Component in "Chapter 5: Developer's Reference"

SpicePack

SpicePack is a collection of COM components that handle common Active Server Pages (ASP) application functionality. These components can be instantiated and called from ASP pages to send and receive e-mail and upload files from client browsers.

See also:

SpicePack Component Reference in "Chapter 5: Developer's Reference"

SQL

SQL, short for Structured Query Language, is the international standard database language used in querying, updating, and managing relational databases. SQL can be used to retrieve, sort, and filter data extracted from a database.

Structured Query Language (SQL)

Structured Query Language (SQL) is the international standard database language used in querying, updating, and managing relational databases. SQL can be used to retrieve, sort, and filter data extracted from a database.

Sun Solaris

Sun Solaris is the computer operating system provided by Sun Microsystems for its family of Scalable Processor Architecture-based processors and for Intel-based processors. Sun emphasizes the system's availability (meaning it seldom crashes), its large number of features, and its Internet-savvy design. Sun developed the platform-independent Java programming language and runtime environment, and Solaris systems include Java and the Java Development Kit (JDK).

Sun provides three extensions for its Solaris operating system:

- Easy Access Server, which is designed to run in a network that also has Windows NT systems

- Enterprise Server, which is aimed at the "business-critical" environment, and includes support for clustering
- Internet Service Provider (Internet service provider) Server

Symbolic link

A symbolic link is a reference to an item that, when accessed, takes the user directly to that item. For example, a symbolic link in one directory (or folder in Windows) could, when double-clicked, open a file that is in a completely different directory. In ASP applications, you can use a symbolic link (also called a virtual link) to redirect the browser to a different HTTP path name than the URL address provided by the user. This function is useful when you want Web site visitors to always use the same URL to get the most current information. A symbolic link can be programmed to refer to any HTTP path name.

System data source name (DSN)

A system data source name (DSN) stores information, in the form of parameters and their values, that the ASP Server needs for connecting to a particular database. The system administrator creates system DSNs by using the Sun Chili!Soft ASP Administration Console. ASP developers can then incorporate this information in a connection string simply by referencing the DSN, rather than specifying all of the parameters in the connection string.

See also:

Enabling Database Connections on the Server in "Chapter 2: Installing and Configuring Sun Chili!Soft ASP"

Configuring a Database in "Chapter 3: Managing Sun Chili!Soft ASP"

Connecting to a Database in "Chapter 4: Building a Sun Chili!Soft ASP Application"

Thread

In computer programming, a thread is a process that is part of a larger process or application. For an application that can handle multiple concurrent users, a thread is the information needed to serve one individual user or a particular service request. It is placeholder information associated with a single use of an application that can handle multiple concurrent users.

Threading refers to the number of processes that are run within a single application. Multi-threading refers to running several processes in rapid sequence within a single program, regardless of which logical method of multi-tasking is being used by the operating system. Because the user's sense of time is much slower than the processing speed of a computer, the impression of multi-tasking appears simultaneous, even though only one task can use a computer processing cycle at a time.

Threading

In computer programming, a thread is a process that is part of a larger process or application. For an application that can handle multiple concurrent users, a thread contains the information needed to serve one individual user or particular service request.

Threading refers to the number of processes that are run within a single application. In multi-threading, a thread is created and maintained for each concurrent user or request from another application. The thread enables the application to know which user is being served as the application is alternately reentered on behalf of different users. Multi-threading refers to running several processes in rapid sequence within a single application, regardless of which logical method of multi-tasking is being used by the operating system.

Multi-threading and multi-tasking are similar and are often confused. Today's computers can only execute one instruction at a time, but because they operate so fast, they appear to run many applications and serve many users simultaneously. The computer operating system gives each application a turn at running, and then requires it to wait while another gets a turn. Each of these applications is viewed by the operating system as a task for which certain resources are identified and tracked. The operating system manages each application on the system (spreadsheet, word processor, Web browser) as a separate task and lets you look at and control items on a task list. If the program initiates an I/O request, such as reading a file or writing to a printer, it creates a thread so that the application will be reentered at the right place when the I/O operation completes. Meanwhile, other concurrent uses of the application are maintained on other threads. Most of today's operating systems provide support for both multi-tasking and multi-threading. They also allow multi-threading within application processes so that the system is saved the overhead of creating a new process for each thread.

The Portable Operating System Interface.4a C specification provides a set of application programming interfaces that enable a programmer to include thread support in the application. Higher-level program development tools and application subsystems and middleware also offer thread management facilities. Object-oriented programming languages also accommodate and encourage multi-threading in several ways. Java supports multi-threading by including synchronization modifiers in the language syntax, by providing class developed for multi-threading that can be inherited by other classes, and by performing background "garbage collection" (recovering data areas that are no longer being used) for multiple threads.

Transition

In object-oriented programming, a transition is a type of filter that marks the passage of a visual object from one style or state to another. There are two kinds of transition filters: **Reveal** and **Blend**. The **Reveal** transition reveals what is behind a visual object by using one of 23 transition styles. The **Blend** transition fades the object in and out of view.

Virtual directory

A virtual directory is a URL defined on a Web server that refers to a physical directory on the server file system. For example, on a Windows-based system, the URL http://myserver/caspdoc might refer to a physical directory having the path name c:\my documents\caspdoc. When a browser requests the URL for a virtual directory, the Web server returns the content contained in the physical directory to which it refers.

Virtual host

A virtual host is a Web server feature that enables one instance of the Web server to service multiple hostnames. Depending on the type of Web server, a virtual host might or might not be given a unique IP address. For more information, consult the documentation for the Web server you are running. Sun Chili!Soft ASP supports virtual hosts.

See also:

Virtual Server

Enabling ASP for a Virtual Host in "Chapter 3: Managing Sun Chili!Soft ASP"

Virtual server

A virtual server is an instance of a Web (HTTP) server that runs on the same physical computer as another instance of the same type of Web server, giving the appearance of being a separate HTTP server. Each virtual server has a unique domain name and IP address.

Virtual servers are frequently used by Internet Service Providers (ISPs) to enable different Web site owners to use and administer their own Web site as though they had complete control of the server. Users of virtual servers can administer their own file directories, add e-mail accounts and address assignments, assign multiple domain names that resolve to a basic domain name without involvement from the ISP, manage their own logs and statistics analysis, and maintain passwords.

See also:

Virtual Host.

Visual Basic (VB)

Visual Basic (VB) is a high-level, visual programming language based on the BASIC (Beginner's All-purpose Symbolic Instruction Code) language, designed by Microsoft for building Windows-based applications. VB was one of the first products to provide a graphical programming environment and a paint metaphor for developing user interfaces. By dragging and dropping controls, such as buttons and dialog boxes, and then defining their appearance and behavior, the VB programmer is able to add a substantial amount of code without getting bogged down in syntactical details.

Although VB is not considered a true object-oriented programming language, it does embrace an object-oriented philosophy. It is sometimes called an event-driven language, because each object can react to different events.

Visual Basic Scripting Edition (VBScript)

Visual Basic Scripting Edition (VBScript) is a scripting language developed by Microsoft that is based on the more complex Visual Basic (VB) programming language. Similar to both JScript and JavaScript, VBScript enables Web authors to include interactive controls, such as buttons and scroll bars, on their Web pages. For a VB programmer, VBScript is the easiest scripting language to learn. (JScript is easier for C/C++ programmers.) Sun Chili!Soft ASP supports both VBScript and JScript.

See also:

VBScript Language Reference in "Chapter 5: Developer's Reference"

Visual InterDev

Visual InterDev is a tool provided by Microsoft for developing HTML-based, data-driven, cross-platform Web applications for the Internet and corporate intranets. Visual InterDev contains a WYSIWYG (What You See Is What You Get) HTML editor, support for Dynamic HTML, client- and server-side debugging, database tools, and support for team-based development. Visual InterDev is also fully interoperable with Microsoft FrontPage for effective workgroup development of Web applications.

W3C

The W3C, short for World Wide Web Consortium, was founded in 1994 to develop common standards and protocols for the World Wide Web. Jointly hosted by the Massachusetts Institute of Technology Laboratory for Computer Science (MIT/LCS) in the United States, the Keio University Shonan Fujisawa Campus in Asia, and the Institut National de Recherche en Informatique et en Automatique (INRIA) in Europe, the W3C is a vendor-neutral international industry consortium. Working with its staff and the global Web community, the W3C produces free, interoperable specifications and sample code, along with reference materials for the World Wide Web.

Web server

A Web server, also called an HTTP server, uses the Hypertext Transfer Protocol (HTTP) to provide information in hypertext format. Clients relay this input from the user to the server and display information on the server in the HTTP format. Other types of Internet-based servers include File Transfer Protocol (FTP) and Gopher. The Web is a network consisting of these types of servers. HTTP servers are commonly referred to as Web servers. The most popular HTTP servers are Apache Web Server, iPlanet Web Server, Enterprise Edition (formerly Netscape), Zeus Web Server, and Microsoft Internet Information Server (IIS).

Web session

Web session defines a period of time during which a user's browser is requesting information from a Web server. Because HTTP is a stateless protocol, it does not provide a mechanism to maintain state information between requests from a browser. With Sun Chili!Soft ASP, developers can use the built-in **Session** object to maintain session information for each user, providing consistent user sessions on the Web.

See also:

ASP Session Object in "Chapter 5: Developer's Reference"

World Wide Web Consortium (W3C)

The World Wide Web Consortium (W3C) was founded in 1994 to develop common standards and protocols for the World Wide Web. Jointly hosted by the Massachusetts Institute of Technology Laboratory for Computer Science (MIT/LCS) in the United States, the Keio University Shonan Fujisawa Campus in Asia, and the Institut National de Recherche en

Informatique et en Automatique (INRIA) in Europe, the W3C is a vendor-neutral international industry consortium. Working with its staff and the global Web community, the W3C produces free, interoperable specifications and sample code, along with reference materials for the World Wide Web.

XML (Extensible Markup Language)

XML, short for Extensible Markup Language, is a simplified subset of Standard Generalized Markup Language (SGML) that provides a file format for representing data, a method for describing data structure, and a mechanism for extending and annotating HTML with semantic information. Allowing an unlimited set of tags, XML tags indicate what kind of data each tag contains, rather than indicating how something should look. For instance, a tag might hold a price, an order number, or a name. The flexibility of XML allows the document's author to determine what kind of data to use and to choose the tag types that most fit the author's needs.

As a universal data format, XML provides a standard for the server-to-server transfer of different types of structured data so that the information can be decoded, manipulated, and displayed consistently and correctly. In addition, it enables the development of three-tier Web applications, acting as the data transfer format between the middle-tier Web server and the client.

XML data type

An XML data type indicates that the contents of an element can be interpreted both as a string and as a typed value (number, date, and so forth). The data type of an XML element indicates that the element contents can be parsed or interpreted to yield an object more specific than a string. Universal Resource Identifiers (URIs) identify data types. The URI is simply a reference to a section of a document that defines the appropriate parser and storage format to the element.

There are two main contexts for data types. The first occurs when dealing with database application-programming interfaces (APIs) in which all elements with the same name typically contain the same type of contents (for example, all sizes contain integers). The second context occurs when the type of content varies widely from instance to instance. The frequency and flexibility of this context varies according to the software being created. For instance, size could contain the integer 6, or the word "small," or even a formula for computing the size.

XML object model

The XML object model tracks the World Wide Web Consortium Document Object Model. The XML parser exposes the XML object model, making it possible to access as objects each of the nodes within an XML tree. Through script, it is then possible to navigate and manipulate an XML tree.

XSL (Extensible Stylesheet Language)

XSL (Extensible Stylesheet Language) is a language that defines the rules for mapping structured XML data and documents. Derived from Document Style Semantics and Specification Language (DSSSL), XSL also has roots in the Standard Generalized Markup Language (SGML) community.

Using XSL, an element can be formatted and displayed in multiple places on a Web page, or rearranged or removed from the page. Developers can then generate a presentation structure that may be quite different from the original data structure. XSL does not replace Cascading Style Sheets (CSS); rather, it is designed to handle the new capabilities of XML that CSS cannot. Although CSS can be used to display simple XML data, CSS is not general enough to handle all of the possibilities generated by XML; the syntax of XSL can.

XSL control

The XSL control is an ActiveX control that enables a Web browser to display output. In other words, the XSL control enables XML data to be displayed within an HTML page by using an XSL style sheet.

Zeus Web Server

The Zeus Web Server is a highly scalable Web server produced by Zeus Technologies.

Index