Sun™ ONE Active Server Pages 4.0

Administrator and Developer's Guide

817-2514-10

Legal Notice

Contents

1 Introduction

Welcome to Sun™ ONE Active Server Pages software, formerly known as Sun Chili!Soft ASP. Sun ONE Active Server Pages 4.0 is Sun's latest ASP engine for the Sun™ ONE Web Server (formerly iPlanet™ Web Server, Enterprise Edition) and the Apache Web Server.

Sun ONE Active Server Pages is a platform-independent implementation of ASP technology, providing:

- An internationalized, cross-platform ASP application environment designed for the seamless deployment of ASP code on a variety of Web servers and platforms.
- A flexible and intuitive Web-based administration console designed to simplify management of ASP applications.
- New XML and Java™ technology extensions designed to extend ASP to other platforms, XML, and Web services.

This chapter provides an introduction to Sun ONE Active Server Pages (also referred to as Sun ONE ASP), and includes a description of what's new and supported in this release, a list of issues you should be aware of before running your installation, an overview of this guide and other helpful resources, and a general introduction to ASP.

Note

The online version of this documentation was developed to meet Section 508 accessibility guidelines. You may wish to adjust text size and other browser settings to suit your personal preferences. To familiarize yourself with this documentation and the information each chapter provides, be sure to review "What the Guide Contains" on page 9.

In this chapter:

New in This Release

This section describes the new features in Sun ONE Active Server Pages 4.0. The features are listed by platform.

UNIX and Linux

Sun ONE Active Server Pages for UNIX and Linux includes the following new features:

- **ASP 3.0 functionality:** Sun ONE ASP includes implementation of the ASP 3.0 functionality listed below (for more information, see "Chapter 9, ASP Built-in Objects Reference" on page 215).
  - **Application.Contents.Remove** (method)
  - **Application.Contents.RemoveAll** (method)
  - **ASPError object** (intrinsic ASP object)
  - **Server.Execute** (method)
  - **Server.GetLastError** (method)
  - **Server.Transfer** (method)
  - **Session.CodePage** (property)
  - **Session.Contents.Remove** (method)
  - **Session.Contents.RemoveAll** (method)
- **VBScript and JScript 5.5 support:** Sun ONE ASP includes support for version 5.5 of Microsoft VBScript and JScript (for more information, see "Chapter 15, Scripting Languages Reference" on page 503).
- **Enhanced language support:** Sun ONE ASP now includes support for English, Dutch, French, German, Japanese Shift-JIS, Simplified Chinese, and Spanish (for more information, see "Configuring International Support" on page 43 and "Developing International Applications" on page 212).
- **Multiple virtual server support:** Sun ONE ASP now supports the multiple virtual server feature of Sun ONE Web Server. This supplements existing support for virtual hosts on the Apache Web server. For related information, see "Defining Applications in a Shared Environment" on page 74 and "Enabling ASP for a Virtual Host" on page 54.
- **Engine deadlock recovery:** Sun ONE ASP provides improved server stability. Failed processes, whether from malformed ASP code or locked database connections, are automatically restarted (for more information, see "Configuring Engine Deadlock Recovery" on page 69).
- **Command-line administration:** In addition to the Sun ONE ASP Administration Console, a browser-based GUI, many administrative tasks can now be performed from the command line (for more information, see "Chapter 5, Command-line Management" on page 83).

- **Database publishing and administration tools:** Sun ONE ASP includes two database tools: Sun ONE ASP Database Publisher (Database Publisher), and Sun ONE ASP Database Management System for MySQL (DBMS).

  Database Publisher is a client/server application that enables a Microsoft Access database running on Windows to be published to a MySQL database running on the UNIX® or Linux platforms (with Sun ONE ASP installed). DBMS is a database administration system for MySQL, enabling MySQL databases to be administered from a user-friendly administration console instead of strictly from the command line. For more information about these tools, see "Database Publisher" on page 135 and "DBMS" on page 148.

- **Enhanced COM-to-Java bridge:** Developers looking to create highly portable ASP applications can integrate cross-platform Java components into their applications directly from ASP scripting (for more information, see "Chapter 12, Chili!Beans Component Reference" on page 465).

- **XML support:** Developers can take advantage of built-in support for XML. Sun ONE ASP enables developers to incorporate pages using the DOM and HTTP features of MSXML 1.0 (Microsoft's XML parser) into their Sun ONE ASP applications, with minimal changes to code (for more information, see "Chapter 13, XML Support" on page 479).

See also:

"Other Features" on page 4

"Supported in This Release" on page 5

"Before You Begin" on page 6

Microsoft Windows NT and Windows 2000

Sun ONE Active Server Pages for Windows includes the following new features:

- **ASP 3.0 functionality** (see "Chapter 9, ASP Built-in Objects Reference" on page 215)
- **VBScript and JScript 5.5 support** (see "Chapter 15, Scripting Languages Reference" on page 503)
- **Multiple virtual server support** (Sun ONE Web Server)
- **Sun ONE ASP Chili!Beans** (see "Chapter 12, Chili!Beans Component Reference" on page 465)
- **Chili!POP3 and Chili!Upload SpicePack components** (see "Chapter 14, SpicePack Component Reference" on page 483)
- **Sun ONE Web Server 6.0 and Apache Web Server 2.0.43 support** (see "Supported in This Release" on page 5)
- **Product updates** (see the note pertaining to Windows in "Checking for Product Updates" on page 28)

See also:

"Before You Begin" on page 6

Other Features

Sun ONE Active Server Pages is designed for the easy setup, administration, and deployment of ASP on the Solaris™, Linux, and Windows platforms. In addition to the features listed in "New in This Release" on page 2, Sun ONE ASP also includes the following:

- **100% pure Active Server Pages support:** Sun ONE ASP supports the most common and frequently used ASP standards, such as ASP 3.0 and version 5.5 of the VBScript and JScript scripting languages. New or existing ASP applications can be deployed with few or no changes to code. For more information, see "Chapter 9, ASP Built-in Objects Reference" on page 215 and "Chapter 15, Scripting Languages Reference" on page 503.
- **Database connectivity tools suite:** Sun ONE ASP includes support for ADO 2.0, and provides DataDirect Connect 4.1 ODBC drivers for all major databases (UNIX and Linux product versions only). Everything needed to integrate data with an ASP application is provided; drivers do not need to be purchased separately.

  For information about the ODBC drivers installed with Sun ONE ASP, see "Supported in This Release" on page 5. For information about configuring the drivers for the data source being used, see "Chapter 6, Configuring a Database" on page 103.
- **Powerful and scalable ASP processing:** Sun ONE ASP includes a browser-based Administration Console for the easy management and configuration of the ASP Server engine (UNIX and Linux product versions only). It also provides server monitoring and diagnostic tools for real-time tracking and monitoring of server performance and availability, as well as ASP error logging for quick detection and diagnosis of server errors. For more information, see "Chapter 2, Using the Administration Console" on page 17 and "Chapter 3, Managing the ASP Server" on page 35.
- **Web application developer tool integration:** Sun works directly with leading Web application development tool vendors such as Macromedia and Adobe® to ensure that applications work seamlessly with Sun ONE ASP.
- **World-class support:** Sun ONE ASP is backed by Sun's global support network and its world-class support. For more information, see "Contacting Customer Support" on page 25.

See also:

"New in This Release" on page 2

"Supported in This Release" on page 5

Supported in This Release

The following table lists the platforms, Web servers, and ODBC-compliant databases supported in this release of Sun ONE Active Server Pages. The ODBC drivers listed in the **Databases** column are installed with Sun ONE ASP.

Version	Platform	Web Servers	Databases
Sun ONE ASP for Solaris	Solaris 8 and 9	- Sun ONE Web Server, Enterprise Edition 6.0 SP5* - Apache 1.3.27 DSO - Apache 2.0.43 DSO *Formerly iPlanet	**DataDirect Connect ODBC 4.1 Wire Protocol drivers** - DB2 Universal Database (UDB) 7.1 - 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 (8.1.7) - Oracle 9i - Sybase Adaptive Server Enterprise 11.9.2 and higher - Sybase Adaptive Server Enterprise 12.5 - Text Files **DataDirect SequeLink 5.3** - Microsoft SQL Server 6.5 - Microsoft Access 2000, 97, and 95 **Open Source** - MySQL 3.23 - PostgreSQL 7.1.3

Version	Platform	Web Servers	Databases
Sun ONE ASP for Linux	- Red Hat Linux 7.3 - SuSE 8.0 Professional **Note:** 2.4 kernel; glibc 2.2.5 Sun ONE ASP may be functional with other Linux distributions supporting the 2.4 kernel and glibc 2.2.5. However, their use is not supported by Customer Support.	- Sun ONE Web Server, Enterprise Edition 6.0 SP5* - Apache 1.3.27 DSO - Apache 2.0.43 DSO - Red Hat Secure Server 7.0 *Formerly iPlanet	**DataDirect Connect ODBC 4.1 Wire Protocol drivers** - DB2 Universal Database (UDB) 7.1 - dBASE 5 - Informix Dynamic Server 2000 (9.20) - Microsoft SQL Server 7.0 SP1 - Microsoft SQL Server 2000 SP1 - Oracle 8i (8.1.7) - Oracle 9i - Sybase Adaptive Server Enterprise 11.9.2 and higher - Sybase Adaptive Server Enterprise 12.5 - Text Files **DataDirect SequeLink 5.3** - Microsoft SQL Server 6.5 - Microsoft Access 2000, 97, and 95 **Open Source:** - MySQL 3.23 - PostgreSQL 7.1.3
Sun ONE ASP for Windows	- Windows NT Server 4.0 SP6 - Windows 2000 Server SP2	- Sun ONE Web Server, Enterprise Edition 6.0 SP5* - Apache 1.3.27 - Apache 2.0.43 *Formerly iPlanet	ODBC drivers are not provided or certified on Windows.

Note

Sun ONE Active Server Pages may function with other versions of the supported Web servers and databases listed in the table above. However, versions not listed have not been certified to run with Sun ONE ASP, and their use is not supported by Customer Support. Zeus Web servers are not supported in this release of Sun ONE ASP.

See also:

"New in This Release" on page 2

"Other Features" on page 4

"Viewing the List of ODBC Drivers" on page 104

Before You Begin

There are certain issues you should be aware of before running your installation of Sun ONE Active Server Pages. This section addresses those issues by platform.

UNIX and Linux

Before running your installation of Sun ONE Active Server Pages for UNIX or Linux, please take note of the following:

- If you chose the default configuration for the Sun ONE ASP Administration Console, the administrator username is configured as "admin" and the password as "root." To protect the security of your server, change the username and password as soon as possible following installation. For more information, see "Configuring Usernames and Passwords" on page 21.
- If you chose not to configure a Web server to run with Sun ONE ASP during installation, you will be prompted to do so the first time you open the Sun ONE ASP Administration Console. For more information, see "Configuring the Web Server after Installation" on page 79.
- Certain Sun ONE ASP settings have important implications for the security of the ASP Server. See "Securing the Server" on page 55 to ensure that the settings are configured appropriately for your specific environment.
- After the installation of Sun ONE ASP, use the diagnostic applications to verify that your installation is functioning correctly. Diagnostics can be accessed from the following URL:

  http://[HOSTNAME]/caspsamp/

  where [HOSTNAME] is the hostname of the Web server configured to run with Sun ONE 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 ONE ASP (/opt/casp by default).

  Be sure to print this information for future reference.

Microsoft Windows NT and Windows 2000

Before running your installation of Sun ONE Active Server Pages for Windows, please take note of the following:

- Sun ONE ASP runs automatically whenever an ASP page is requested by a user (provided the Web server is running). Sun ONE ASP runs until the Web server is stopped. When the Web server is restarted, Sun ONE ASP will not run until an ASP page is requested by a user.
- The Sun ONE ASP Administration Console is referenced throughout this documentation. The Administration Console is available for the UNIX and Linux versions of the product only. With Sun ONE ASP for Windows, all configuration is performed during installation. Some of the configuration information is stored in registry settings, however, and expert users can use regedit to edit those settings. For more information, see "Editing the Windows Registry" on page 515.

- With Sun ONE ASP for Windows, ASP applications are defined by adding aliases or virtual directories to the Web server. Sun ONE ASP treats each alias or virtual directory as an ASP application. With Sun ONE Web Server, ASP applications are defined by adding an "additional document directory" using the server's Administration tool. With the Apache Web server, ASP applications are defined by adding an alias to the httpd.conf file.
- ODBC drivers are not installed with Sun ONE 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. Consult Microsoft documentation for more information.

See also:

"Defining ASP Applications (ASP Server)" on page 46

"Creating Database Connections (ASP Server)" on page 44

About This Guide

This guide provides information about the configuration and use of Sun ONE Active Server Pages. It also provides basic information about building ASP applications, as well as developer reference information.

Two versions of the guide are included with the product: one in HTML format optimized for online viewing, and one in Adobe® PDF format optimized for printing. 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

We would like to know what you found useful in this documentation, and what you think could be improved. Please use the form at the following URL to provide your comments:

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

Note

The online version of this documentation was developed to meet Section 508 accessibility guidelines. You may wish to adjust text size and other browser settings to suit your personal preferences. To familiarize yourself with this documentation and the information each chapter provides, be sure to review "What the Guide Contains" on page 9.

What the Guide Contains

The following table describes the contents of each chapter in this guide, and the users who will benefit most from reading them. The first half of the guide contains information about the configuration and use of Sun ONE Active Server Pages. The second half provides developer reference information (starting with "ASP Built-in Objects Reference").

Note

This guide does not include installation instructions. Detailed installation and getting started information can be found in the QuickStart Guide included with Sun ONE ASP. The most current application notes are found in the README file included with the product, and you should review the README before using Sun ONE ASP. For more information about accessing documentation resources, see "How the Guide is Accessed" on page 11.

Chapter	Audience	Description
"Chapter 1, Introduction" on page 1	System administrators and Web developers	Provides an overview of Sun ONE ASP and this documentation, including information about new features, supported platforms, and steps you should take before running your installation of Sun ONE ASP.
"Chapter 2, Using the Administration Console" on page 17	System administrators	Describes how to access and use the Sun ONE ASP Administration Console for basic tasks such as configuring usernames and passwords, contacting Customer Support, and viewing the README. The Administration Console is the browser-based application used for managing Sun ONE ASP.
"Chapter 3, Managing the ASP Server" on page 35	System administrators	Provides information about administering Sun ONE ASP from the Sun ONE ASP Administration Console, including information about changing ASP Server configuration settings, configuring security, and optimizing server performance. Also provides overview information about creating database connections and enabling users to publish ASP applications to the Web server.
"Chapter 4, Managing the Web Server" on page 77	System administrators	Provides information about managing certain Web server settings from the Sun ONE ASP Administration Console (note that most Web server management is handled through the Web server's own management interface).
"Chapter 5, Command-line Management" on page 83	System administrators	Provides information about performing management tasks from the command line.

Chapter	Audience	Description
"Chapter 6, Configuring a Database" on page 103	System administrators	Describes how to create and edit Data Source Names (DSNs), and how to configure the ASP Server to connect with supported databases.
"Chapter 7, Using Database Tools" on page 135	System administrators and Web developers	Describes the Sun ONE ASP Database Publisher and DBMS tools. Database Publisher enables a Microsoft Access database running on Windows to be published to a MySQL database running on UNIX or Linux (with Sun ONE ASP installed). DBMS enables MySQL databases to be administered from an administration console instead of strictly from the command line.
"Chapter 8, Building Sun ONE ASP Applications" on page 181	System administrators and Web developers	Introduces the basics of developing ASP applications, including information about creating an ASP page, adding scripts and server-side includes, defining the application on the server, and developing international applications. Also discusses how to extend ASP applications by using objects and components, how to connect to a database, and how to publish a Sun ONE ASP application.
"Chapter 9, ASP Built-in Objects Reference" on page 215	Web developers (reference chapter)	Provides reference information about built-in (intrinsic) ASP objects.
"Chapter 10, ASP Component Reference" on page 271	Web developers (reference chapter)	Provides reference information about ASP components.
"Chapter 11, ADO Component Reference" on page 301	Web developers (reference chapter)	Provides reference information about ADO (ActiveX Data Objects).
"Chapter 12, Chili!Beans Component Reference" on page 465	Web developers (reference chapter)	Provides reference information about the Sun ONE ASP Chili!Beans ActiveX control, a wrapper that enables Java objects to be used by COM controllers. Also provides reference information about the new ASP servlet interface, which enables Java objects designed for use in JSPs to be integrated into Sun ONE ASP applications directly from ASP scripting.
"Chapter 13, XML Support" on page 479	Web developer (reference chapter)	Provides reference information about XML support provided in Sun ONE ASP.
"Chapter 14, SpicePack Component Reference" on page 483	Web developers (reference chapter)	Provides reference information about the Sun ONE ASP SpicePack components, a set of COM components that handle commonly used ASP application functionality.

Chapter	Audience	Description
"Chapter 15, Scripting Languages Reference" on page 503	Web developers (reference chapter)	Provides reference information about Microsoft VBScript and JScript, and about Sun ONE ASP VBScript and Sun ONE ASP JavaScript (the Sun ONE ASP scripting engines).
"Appendix A, Errors Reference" on page 505	System administrators and Web developers	Explains error messages you might encounter when using Sun ONE ASP.
"Appendix B, Troubleshooting" on page 513	System administrators and Web developers	Provides access to troubleshooting information, most of which is available from the Sun ONE ASP knowledge base.
"Appendix C, Advanced Administration Options" on page 515	System administrators and Web developers	Describes advanced administration options for expert users of Sun ONE ASP.
"Glossaries" on page 533	System administrators and Web developers	Contains two glossaries: A general glossary with terms you might encounter when administering Sun ONE ASP and developing ASP applications, and a user interface glossary with terms specific to the Sun ONE ASP Administration Console GUI.

See also:

"How the Guide is Accessed" on page 11

"Other Resources" on page 13

How the Guide is Accessed

Sun ONE Active Server Pages documentation can be accessed in a number of ways:

- From the Sun ONE ASP Administration Console, as described in "Accessing Product Documentation" on page 23 (UNIX and Linux only).
- From your Web server, if you chose the option to install documentation during installation. With your Web server running, go to:

  http://[HOSTNAME]/caspdoc/ (for online documentation)

  - or -

  http://[HOSTNAME]/caspdoc/pdf/SunONEActiveServerPages4.pdf (for the PDF)

  where [HOSTNAME] is the hostname of the Web server configured to run with Sun ONE ASP.
- From the Sun ONE ASP developer Web site at:

  http://developer.chilisoft.com/caspdoc/
- From the Sun ONE ASP product home page. With your Web server running, go to:

  http://[HOSTNAME]/caspsamp

where [HOSTNAME] is the hostname of the Web server configured to run with Sun ONE ASP (see "Product Home Page" on page 13).

In addition to the complete product documentation, two other Sun ONE ASP documentation resources will also be helpful to you:

- The QuickStart Guide, a text file that provides installation and migration instructions, and information about getting started with Sun ONE ASP. The guide is included on the installation CD-ROM. If you downloaded Sun ONE ASP from the Web, the guide is included in the installation files extracted from the tar file. The QuickStart Guide can also be accessed from the Sun ONE ASP developer Web site at:

  http://developer.chilisoft.com/caspdoc/

- The README, a text file that contains the latest product information and application notes. The README is included with the product and can be accessed as described in "Viewing the README File" on page 24. The most current README can be accessed from the Sun ONE ASP developer Web site at the URL listed above.

See also:

"What the Guide Contains" on page 9

"Other Resources" on page 13

Guide Conventions

The following table lists the typographic conventions used in this guide.

Convention	Used for
Bold	- User interface elements - Objects, methods, properties - Emphasis
`Monospace font`	- Code - Command-line commands that must be typed exactly as shown
Italics	- Placeholders representing conditional information that must be inserted (such as <*install directory*>) - Parameters - Variables - Emphasis

See also:

"About This Guide" on page 8

"What the Guide Contains" on page 9

"How the Guide is Accessed" on page 11

Other Resources

In addition to the product documentation, the resources listed below will also be helpful to you.

Product Home Page

The Sun ONE Active Server Pages product home page (referred to as the Start Page in earlier releases) provides links to resources that will help you get the most out of Sun ONE ASP. The page provides access to diagnostics, the 10-step Tour, product documentation, developer references, and Customer Support. The product home page can be accessed from:

- http://[HOSTNAME]/caspsamp

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

- The following URL:

  http://developer.chilisoft.com/caspsamp

- The **Product Home Page** link at the top of each page in the online version of the Sun ONE ASP product documentation.

Note

To use ASP functionality in diagnostics and the 10-step Tour, **Allow session state** must be set to **yes** on the **Server Settings** page in the Sun ONE ASP Administration Console. For more information about this setting, see "Enabling Session State" on page 42. For Windows systems, see "Editing the Windows Registry" on page 515.

"Diagnostic Applications" on page 13

"Knowledge Base" on page 14

"Support Forum" on page 15

"Developer Web Site" on page 15

Diagnostic Applications

Diagnostic applications are used to verify that your ASP environment is working correctly. Following installation, and with your Web server running, diagnostic applications can be accessed from:

http://[HOSTNAME]/caspsamp/

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

The following tables lists the diagnostic applications installed with Sun ONE Active Server Pages.

Diagnostic	Used for
HELLO	Tests the functionality of ASP and VBScript by using a simple "Hello World" script.
JSCRIPT	Tests the functionality of JScript.
SERVER	Tests the ASP Server-to-Web server connection by retrieving the standard Web server variables.
COMPONENTS	Tests the functionality of additional components installed with Sun ONE ASP.
ADO	Accesses ADO from VBScript. Connects to a sample database by using ODBC to test the functionality of ADO and the dBASE ODBC driver.
JSADO	Performs the same test as the ADO diagnostic, but accesses ADO from JScript rather than from VBScript.
SQLEXECUTE	Uses ADO to execute a SQL statement and display the results. To use this application, you must first create a system DSN for your database on the ASP Server (see "Adding a DSN" on page 106).

Note

To use ASP functionality in the diagnostic applications, **Allow session state** must be set to **yes** on the **Server Settings** page in the Sun ONE ASP Administration Console. For more information about this setting, see "Enabling Session State" on page 42. For Windows systems, see "Editing the Windows Registry" on page 515.

See also:

"Product Home Page" on page 13

"Other Resources" on page 13

Knowledge Base

The Sun ONE Active Server Pages knowledge base is a valuable technical resource, providing troubleshooting information, answers to frequently asked questions, and useful "how-to" tips. To access the knowledge base, go to:

http://developer.chilisoft.com/kb/

See also:

"Other Resources" on page 13

Support Forum

The Sun ONE Active Server Pages Support Forum is a great way to interact with other members of the Sun ONE ASP community, sharing tips, experiences, and expertise. To access the Support Forum, go to:

http://developer.chilisoft.com/forum/

See also:

"Other Resources" on page 13

Developer Web Site

The Sun ONE Active Server Pages developer Web site provides many helpful development resources, including sample ASP applications that demonstrate the basics of building Sun ONE ASP applications. To access the developer Web site, go to:

http://developer.chilisoft.com

See also:

"Other Resources" on page 13

About ASP

Active Server Pages (ASP) is a specification for a dynamically created Web page with 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. Sun ONE Active Server Pages is a platform-independent implementation of ASP technology, enabling ASP applications to be run on a variety of Web servers running under UNIX, Linux, and Windows.

An ASP application consists of ASP pages published on a Web site. An ASP page is simply a plain text file with the .asp file name extension. ASP pages can contain HTML code, client-side scripts, and server-side scripts. A Sun ONE ASP page uses VBScript or JScript code to access the ASP object model, which exposes functionality often used in Web application environments. When a user requests an ASP page, the Web server passes execution to the Sun ONE ASP Server, which processes the scripts, generates an HTML page, and sends it back to the browser.

Benefits of ASP

ASP was designed as a faster and easier alternative to CGI scripting using Perl or C. It provides an easy-to-learn scripting interface (including native support for 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. ADO components can be used to perform additional

functions, including accessing ODBC-compliant databases, and outputting data to text files. Java components and XML can be used to extend ASP scripts.

Additional benefits include the following:

- 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, allowing programmers and Web designers to focus exclusively on what they do best.
- Server-side ASP scripts can be used to store HTML form information in a database, personalize Web sites according to visitor preferences, or use different HTML features based on the 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.
- For the HTML author, ASP is an easy way to begin creating Web applications. To process user input on the Web server with CGI applications, a programming language such as Perl or C must be learned. With ASP, however, you can collect HTML form information and pass it to a database by using simple server-side scripts written in VBScript of JScript that are embedded directly in your HTML documents.
- ASP is language-neutral, so if you're skilled at a scripting language such as VBScript or JScript, you already know how to use ASP.
- If you develop Web applications by using a programming language such as Java, Visual Basic, or C++, you will appreciate the flexibility of ASP. In addition to 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.

2 Using the Administration Console

The Sun ONE Active Server Pages Administration Console is a browser-based application used for managing Sun ONE ASP. It enables administrators to configure and control the Sun ONE ASP Server and its bindings to Web servers and database servers from a Web browser, either locally or remotely.

Most product configuration settings are accessible from the Administration Console. Whenever possible, the Administration Console should be used for product configuration.

This chapter describes how to access and use the Administration Console for basic tasks. Subsequent sections describe the use of the Administration Console for server management and configuration.

Note

Expert users can also perform certain tasks from the command line. For more information, see "Chapter 5, Command-line Management" on page 83.

In this chapter:

See also:

Accessing the Administration Console

The Sun ONE Active Server Pages 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 the computer running Sun ONE ASP is started.

To access the Administration Console, you must know the hostname of the Web server configured to run with Sun ONE ASP, and the port on which the Administration Console is running (5100 by default).

If you did not configure a Web server to run with Sun ONE ASP during installation, you will be prompted to do so the first time you open the Administration Console.

To access the Administration Console

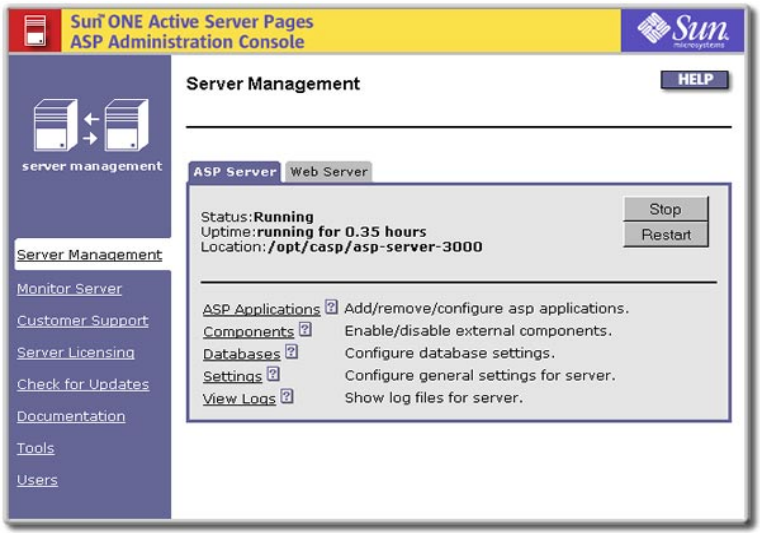
1. In your browser address bar, enter the following URL:

   http://[HOSTNAME]:[PORT]

   where [HOSTNAME] is the hostname of the Web server configured to run with Sun ONE ASP, and [PORT] is the port on which the Administration Console is running (5100 by default).

2. When prompted, type the username and password specified during installation. If you chose the default configuration for the Administration Console during installation, the username is configured as "admin" and the password as "root." You should change these from the defaults as soon as possible, as described in "Configuring Usernames and Passwords" on page 21

After typing the username and password, the Administration Console opens and displays the **Server Management** page. This page is used to access configuration settings for Sun ONE ASP, view information about the installation, and start and stop the associated Web server.



Note

When Sun ONE ASP is installed and your Web server is running, you can also access the Administration Console from the Sun ONE ASP product home page at:

http://[HOSTNAME]/caspsamp/

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

To access the Sun ONE ASP Administration Console using a URL, you must specify its port number. While the default port number is 5100, this number might be different if 5100 was already in use when you installed Sun ONE ASP, or if you specified a different port number for the Administration Web site during installation. If you don't know the correct port number, this information can be found in the Sun ONE ASP installation summary file. This file is named install_summary.

On UNIX and Linux platforms, the installation summary file is found in the following location:

/[C-ASP_INSTALL_DIR]/logs/

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

The entry reads as follows:

```
Administration console installed:
URL: http://[YOUR_WEB_SERVER_HOSTNAME]:[PORT_NUMBER]
Port: [PORT_NUMBER}
```

Starting and Stopping the Administration Web Server

An administration Web server is installed with Sun ONE Active Server Pages, and hosts the Sun ONE ASP Administration Console. By default, the administration Web server is configured to start when the computer running Sun ONE ASP is started. Although you can start and stop the ASP Server by using the Administration Console (as described in "Stopping and Restarting the ASP Server (Admin Console)" on page 41), to start or stop the administration Web server, you must use the command-line utility, `admtool`, which is installed with Sun ONE ASP.

To start or stop the administration Web server

1. Telnet or log in to the computer running Sun ONE ASP as root.
2. Change directories (cd) to the root installation directory (/opt/casp by default).
3. Start the `admtool` utility with 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**).

Configuring Usernames and Passwords

During installation a username and password is created to restrict access to the Sun ONE ASP Administration Console. The Administration Console can be used to add, edit, and delete usernames and passwords as follows:

- Any administrator can add or delete any other administrator user.
- Any administrator can change his or her own password, but not that of any other administrator.

Caution

If you chose the default configuration for the Administration Console during installation, the administrator username is configured as "admin" and the password as "root." To protect the security of your server, you should change these from the defaults as soon as possible.

To configure usernames and passwords

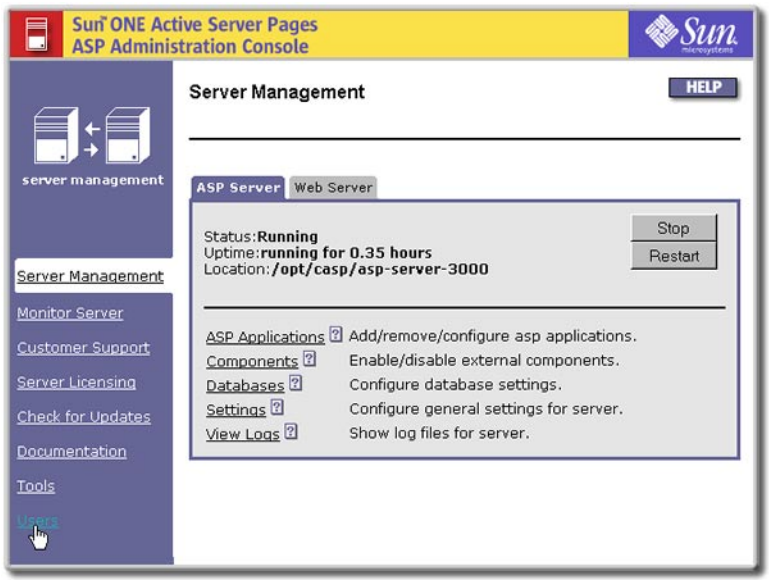
1. Open the Administration Console by using the following URL:

   http://[HOSTNAME]:[PORT]

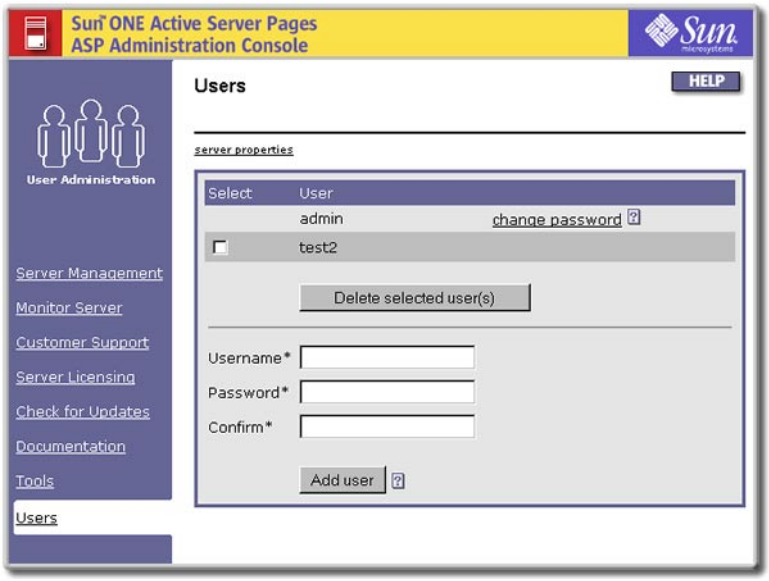
   where [HOSTNAME] is the hostname of the Web server configured to run with Sun ONE ASP, and [PORT] is the port on which the Administration Console is running (5100 by default).

   The **Server Management** page displays.

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



   The **Users** page displays.



3. Perform the desired action:

   - To change your password, click **change password**. In the **Change Password** dialog that displays, provide password information, and then click **OK**. You can change your own password, but not that of any other user.

     - or -

   - To add a user, specify the username, password, and password confirmation in the corresponding boxes on the **Users** screen, and then click **Add user**.

     - or -

   - To delete a user, select the check box that corresponds to the user and click **Delete selected user(s)**.

Note

It is not necessary to restart the ASP Server after making these changes.

See also:

"Accessing the Administration Console" on page 18

Accessing Product Documentation

As described in "How the Guide is Accessed" on page 11, there are several ways to access product documentation. This section describes how to access documentation from the Sun ONE ASP Administration Console.

To access documentation from the Administration Console

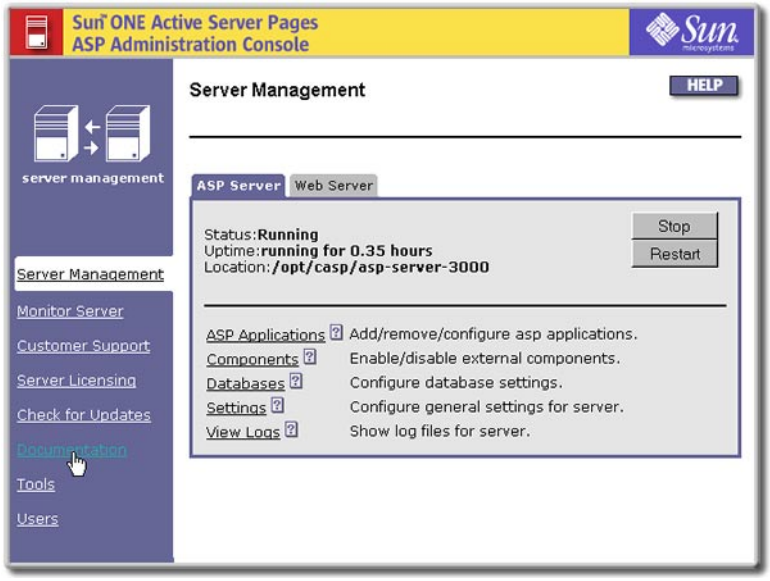
1. Open the Administration Console by using the following URL:

   http://[HOSTNAME]:[PORT]

   where [HOSTNAME] is the hostname of the Web server configured to run with Sun ONE ASP, 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**.



   Online documentation displays.

See also:

"What the Guide Contains" on page 9

"Viewing the README File" on page 24

"Other Resources" on page 13

Viewing the README File

A README file is installed on your computer during the installation of Sun ONE Active Server Pages. The README file provides the most current product information and application notes. The README file can be accessed from the Sun ONE ASP Administration Console, as described in the following procedure. The README file can also be found in the following directory:

/[C-ASP_INSTALL_DIR]/

where [C-ASP_INSTALL_DIR] is the path name of the Sun ONE ASP installation directory (/opt/casp by default on UNIX and Linux).

The most current README file can also be found on the Sun ONE ASP developer Web site at the following URL:

http://developer.chilisoft.com/caspdoc/

Note

In addition to the README file and product documentation, a number of other resources are available to assist you with the use and configuration of Sun ONE ASP. For more information, see "Other Resources" on page 13.

To view the README file

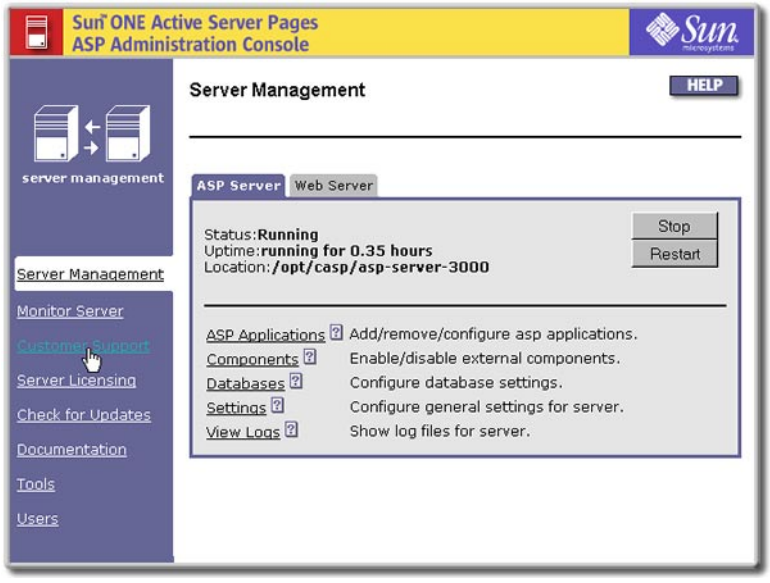
1. Open the Administration Console by using the following URL:

   http://[HOSTNAME]:[PORT]

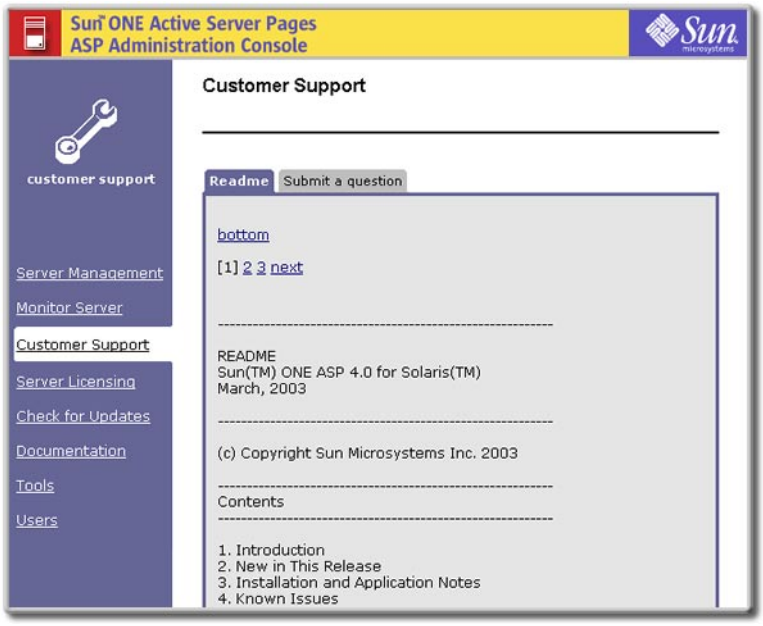
   where [HOSTNAME] is the hostname of the Web server configured to run with Sun ONE ASP, and [PORT] is the port on which the Administration Console is running (5100 by default).

   The **Server Management** page displays.

2. In the left navigation pane, click **Customer Support**.



The README file displays.



Contacting Customer Support

Use the following procedure to contact Customer Support if you encounter problems while using Sun ONE Active Server Pages.

To contact Customer Support

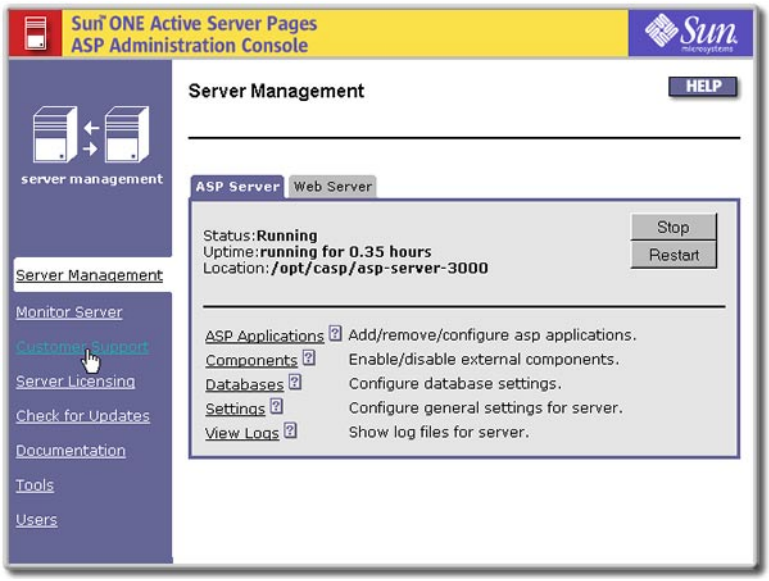
1. Open the Administration Console by using the following URL:

   http://[HOSTNAME]:[PORT]

   where [HOSTNAME] is the hostname of the Web server configured to run with Sun ONE ASP, and [PORT] is the port on which the Administration Console is running (5100 by default).

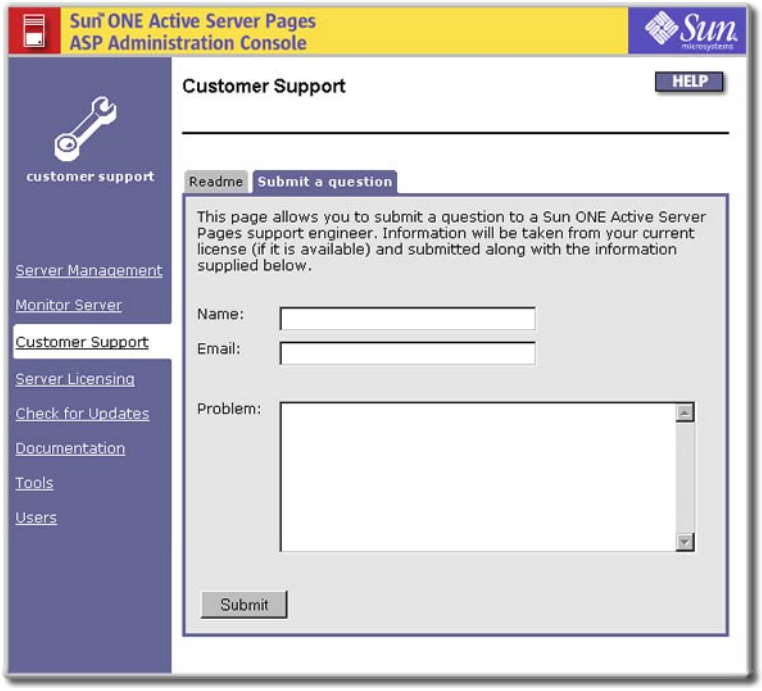
   The **Server Management** page displays.

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**.

Note

If you are unable to submit your problem as described in the previous steps, contact Customer Support using the Web form at the following URL:

http://developer.chilisoft.com/support/supportrequest.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 ONE ASP Administration Console.

Installing a New Serial Number

There may be times when you need to install a new serial number, such as when you're upgrading your product license. This serial number is provided by Sun.

To install a new serial number

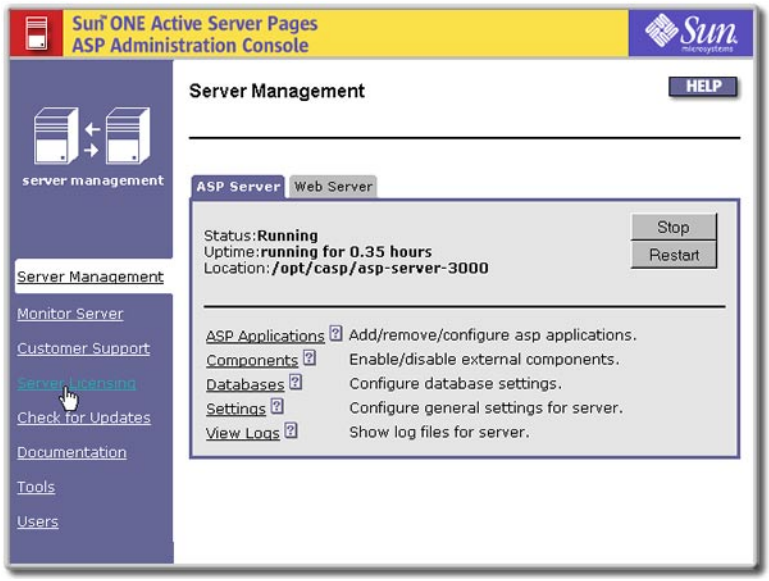
1. Open the Administration Console by using the following URL:

   http://[HOSTNAME]:[PORT]

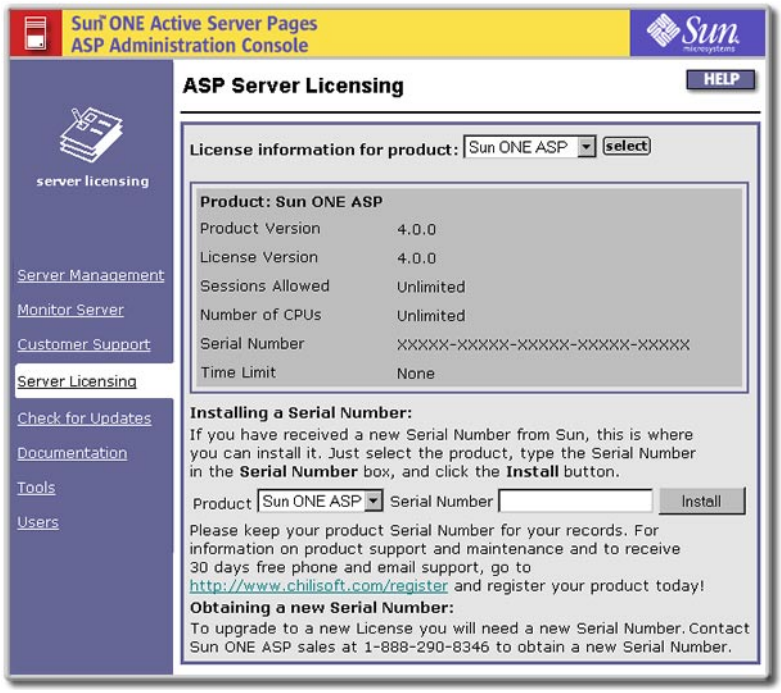
   where [HOSTNAME] is the hostname of the Web server configured to run with Sun ONE ASP, and [PORT] is the port on which the Administration Console is running (5100 by default).

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

Note

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

Checking for Product Updates

Product updates and fixes are occasionally provided to enhance the security and performance of Sun ONE Active Server Pages software. When using the Sun ONE ASP Administration Console (UNIX and Linux versions), you will see periodic prompts asking if you want to check for updates. You can also check for updates on demand, to quickly determine if your specific installation of Sun ONE ASP is current. Use the following procedure to check for updates on demand from the Sun ONE ASP Administration Console.

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

Note

You can also check for product updates with Sun ONE Active Server Pages for Windows. To do so, go to the Sun ONE ASP product Web page and click

Check For Updates. The product page can be accessed from **Start** > **Sun ONE ASP** > **Samples**.

To check for product updates

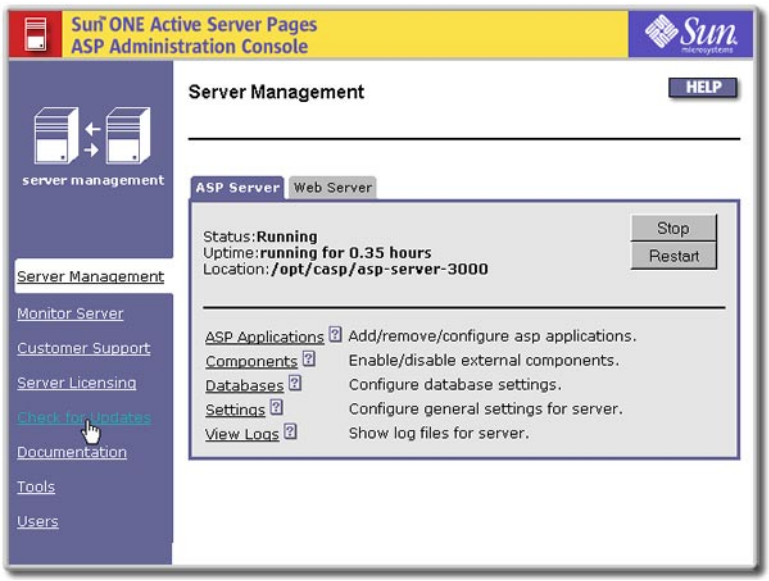
1. Open the Administration Console by using the following URL:

   http://[HOSTNAME]:[PORT]

   where [HOSTNAME] is the hostname of the Web server configured to run with Sun ONE ASP, 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 ONE ASP Product Update** page displays, listing information about your Sun ONE ASP installation.

3. Make your desired selection:

   - Select **Check for update now** to check our 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.

Enabling External Components

The Sun ONE ASP Administration Console provides access to external Sun ONE ASP SpicePack and Chili!Beans components. The SpicePack is a set of 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.

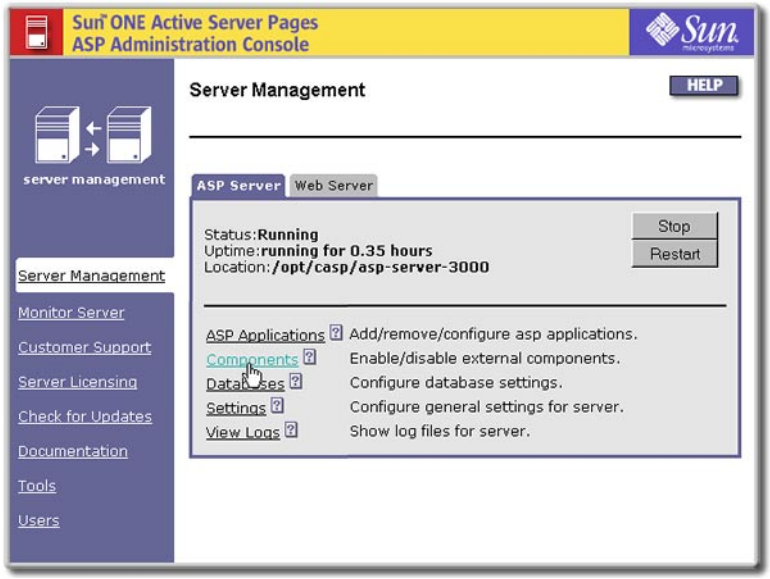
To enable external components

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

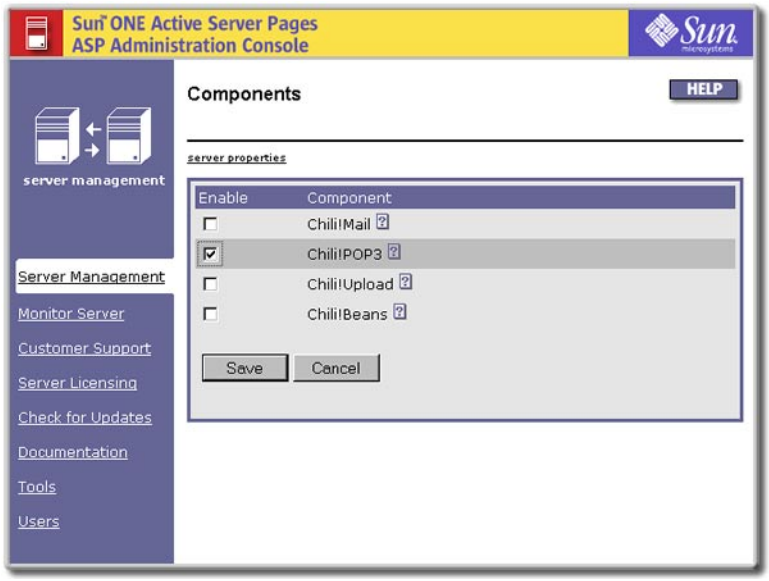
   http://[HOSTNAME]:[PORT]

   where [HOSTNAME] is the hostname of the Web server configured to run with Sun ONE ASP, 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 (for specific information about these components and any additional settings, see the "See also" references listed after this procedure).
4. When finished, click **Save** to save your changes, or **Cancel** to revert to the settings that were last saved.

   The **Server Management** page displays.
5. If you changed the status of the Chili!Beans or Chili!Upload components, 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 Chili!Mail or Chili!POP3 components.

Note

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

See also:

"Enabling SpicePack Components" on page 483

"SpicePack Component Reference" on page 483

"Enabling Chili!Beans" on page 466

"Chili!Beans Component Reference" on page 465

Enabling Database Tools

Sun ONE Active Server Pages includes two database tools: Sun ONE ASP Database Publisher (Database Publisher), and Sun ONE ASP Database Management System (DBMS).

Database Publisher is a client/server application that enables a Microsoft Access database running on Windows to be published to a MySQL database running on UNIX or Linux (with Sun ONE ASP installed). DBMS is a database administration system for MySQL, enabling MySQL databases to be administered from a user-friendly administration console instead of strictly from the command line.

Both Database Publisher and DBMS are enabled and administered from the **Tools** page in the Sun ONE ASP Administration Console.

To enable or disable database tools

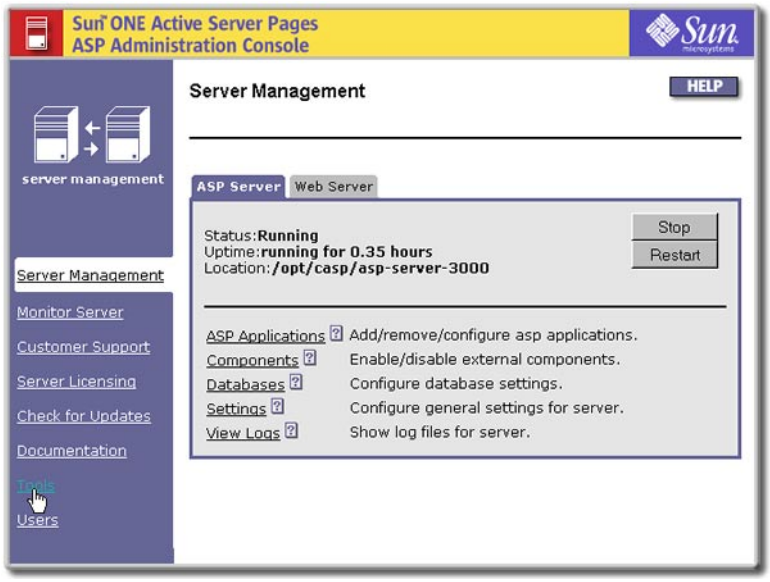
1. Open the Administration Console by using the following URL:

   http://[HOSTNAME]:[PORT]

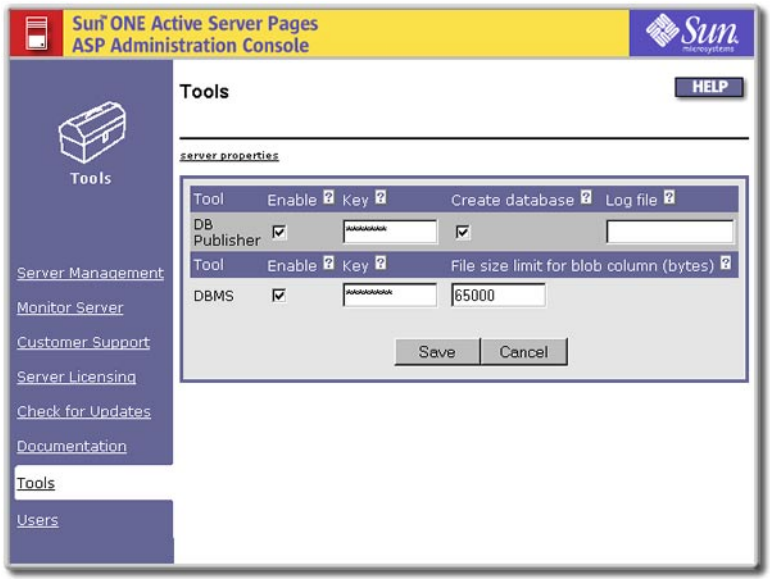
   where [HOSTNAME] is the hostname of the Web server configured to run with Sun ONE ASP, and [PORT] is the port on which the Administration Console is running (5100 by default).

   The **Server Management** page displays.

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



   The **Tools** page displays.



3. Click the **Enable** check box to select or clear (enable or disable) the tools as desired.
4. When finished, click **Save** to save your changes, or **Cancel** to revert to the settings that were last saved.

   It is not necessary to restart the ASP Server.

Note

For information about the administration and use of these tools, see "Database Publisher" on page 135 and "DBMS" on page 148.

3 Managing the ASP Server

Sun ONE Active Server Pages includes an ASP Server that processes ASP page requests. The ASP Server is managed from the Sun ONE ASP Administration Console. Most configuration settings are accessible from the Administration Console, and it is strongly recommended that the Administration Console be used for product configuration.

This chapter describes how to manage the ASP Server from the Sun ONE ASP Administration Console.

Note

While it is strongly recommended that the Administration Console be used for product configuration, expert users can also perform certain tasks from the command line. For more information, see "Chapter 5, Command-line Management" on page 83.

In this chapter:

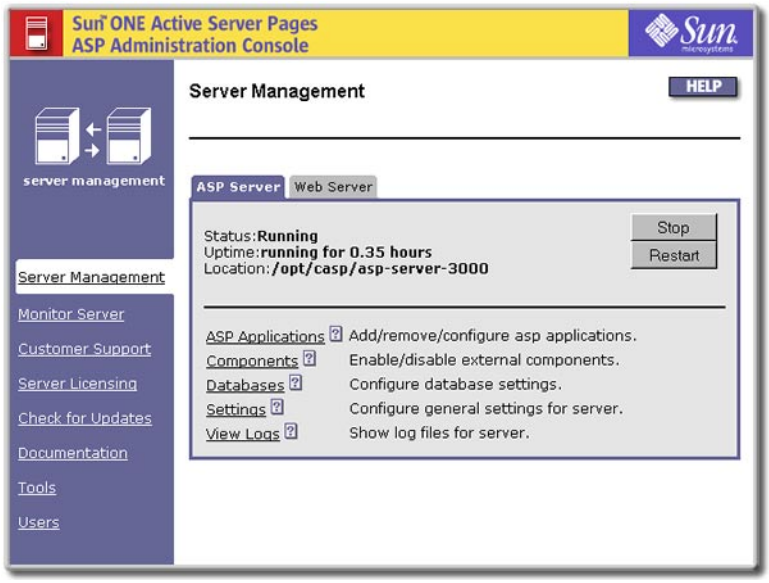
See also:

"Chapter 4, Managing the Web Server" on page 77

Server Management Overview (ASP)

The ASP Server is managed from the **Server Management** page in the Sun ONE Active Server Pages Administration Console. This page has two tabs, **ASP Server** and **Web Server**, which are used to access settings for the ASP Server and the Web server. This section discusses basic management of the ASP Server. For information about managing the Web server, see "Chapter 4, Managing the Web Server" on page 77.

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



The following items are displayed on the **ASP Server** tab:

- **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 (Admin Console)" on page 41.
- The **ASP Applications** link displays settings for adding, removing, and configuring ASP applications. For more information, see "Configuring ASP Applications" on page 47.
- The **Components** link provides access to settings for the Sun ONE ASP SpicePack and Chili!Beans components, and displays the page where the components are enabled or disabled. For more information, see "Enabling External Components" on page 30.
- The **Databases** link displays database configuration settings. For more information, see "Chapter 6, Configuring a Database" on page 103.

- The **Settings** link displays general ASP Server settings. For more information, see "Changing ASP Server Settings" on page 37.
- The **View Logs** link provides access to pages from which the log files enabled for the ASP Server can be viewed. For more information, see "Viewing the ASP Errors Log" on page 64.

Note

While most settings for Sun ONE ASP should be configured in the Administration Console, expert users can perform certain tasks from the command line. For more information, see "Chapter 5, Command-line Management" on page 83.

See also:

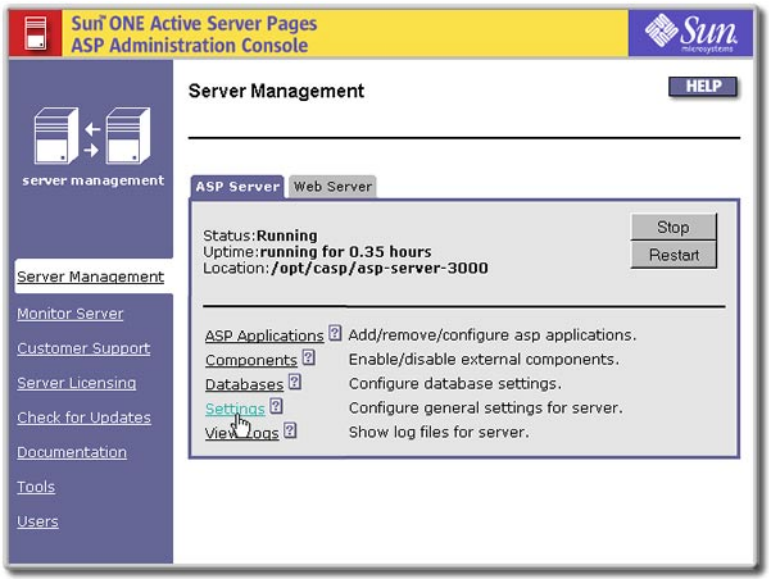
"Server Management Overview (Web)" on page 77

Changing ASP Server Settings

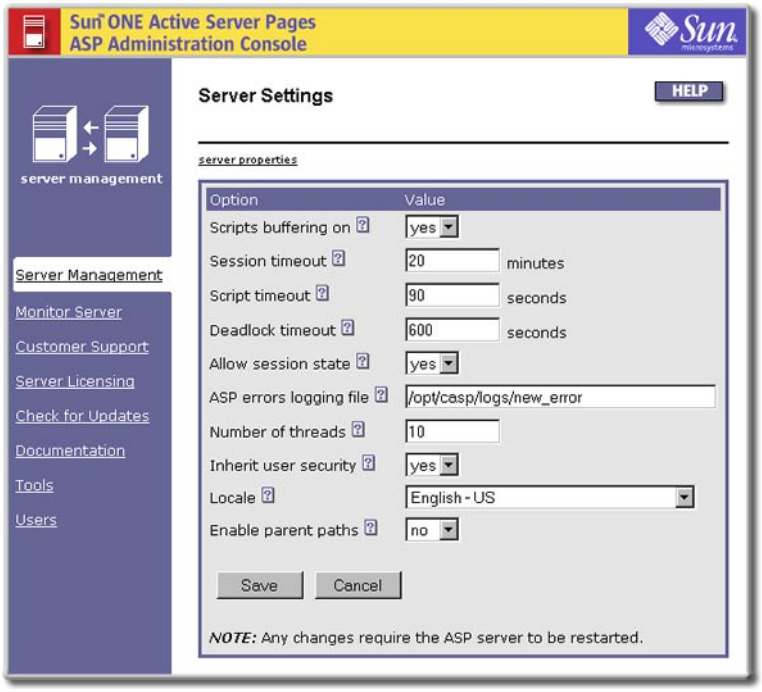
The Sun ONE Active Server Pages Administration Console **Server Settings** page provides access to the basic configuration settings for the ASP Server. The following procedure describes how to change server settings. To put any changes into effect, you must restart the ASP Server.

To change ASP Server settings

1. Open the Administration Console (see "Accessing the Administration Console" on page 18).
2. On the **ASP Server** tab of the **Server Management** page, click **Settings**.



The **Server Settings** page displays.



3. Configure the settings as desired (settings are described in the following table).
4. When finished, click **Save** to save your changes, or **Cancel** to revert to the settings that were last 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.

The following table describes ASP Server settings.

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, which makes debugging easier. This setting is **yes** by default. For more information, see "Enabling Scripts Buffering" on page 66.

Setting	Explanation
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" on page 67.
Script timeout	This specifies the number of seconds the ASP Server waits for a page to finish processing before canceling the page request. A value for ScriptTimeout specified in a script will always override this value. This setting is **90** seconds by default. For more information, see "Changing the Script Timeout Value" on page 68. **Note:** If the deadlock timeout (below) is set to a value lower than the script timeout, the ASP engine will restart once the time specified for the deadlock timeout has elapsed.
Deadlock timeout	This specifies the number of seconds that should elapse before the ASP Server is considered deadlocked and the engine is restarted. This setting is **600** seconds (10 minutes) by default. For more information, see "Configuring Engine Deadlock Recovery" on page 69. **Note:** If the deadlock timeout is set to a value lower than the script timeout (above), the ASP engine will restart once the time specified for the deadlock timeout has elapsed.
Allow session state	This specifies whether the ASP Server maintains session state. This setting must be enabled (**yes**) in order for **Session** objects in scripts to function. This setting is **yes** by default. For more information, see "Enabling Session State" on page 42.
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 ONE 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 Errors Logging" on page 63.
Number of threads	This specifies the number of threads the ASP Server handles simultaneously. The default is **5**. 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. A maximum number of up to 20 threads is recommended. DO NOT set this to a number greater than 20. For more information, see "Configuring Multi-threading" on page 71.

Setting	Explanation
Inherit user security	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, especially if you are running Sun ONE Web Server. When **Inherit user security** is set to **yes** (Inherit User Security mode), 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 ONE ASP and is available only for Sun ONE ASP running with the Apache Web server (for Sun ONE Web Server, see the following discussion of the Defined User Security mode). When **Inherit user security** is set to **no** (Defined User Security mode), the ASP Server runs with the permissions of the user who started the ASP Server, unless a different user and group is specified in the Sun ONE ASP configuration file (casp.cnfg). This can create a security risk for your server. If **Inherit user security** is set to **no** (or if you are running Sun ONE Web server), be sure to define a user and group in casp.cnfg, as described in "Editing the Sun ONE ASP Configuration File" on page 517 (see the `[default machine]` keyword). The ASP Server will then run with the permissions of that user and group. Defined User Security mode is available for Sun ONE ASP running with both the Apache and Sun ONE Web servers. For more information about the security modes and their implications, see "Setting the Security Mode" on page 57. **Note:** ADO logging will not be functional if **Inherit user security** is set to **yes**. Also note that the **Inherit user security** setting does not add any restrictions to executing Java code. For example, if you want to restrict Java code to access files within the application directory, the proper permissions should be in the bean.policy file.
Locale	This specifies the locale setting. The ASP Server uses the appropriate code page for the language associated with the locale specified. It also correctly formats dates, numbers, and currency according to the locale. For more information, see "Configuring International Support" on page 43. Supported locales vary by platform.

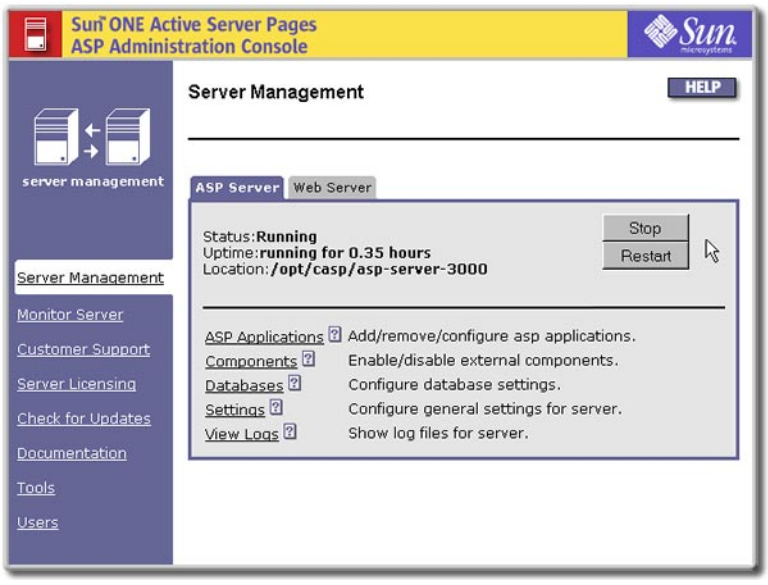
Setting	Explanation
Enable parent paths	This enables file system access by an ASP application to a directory in the file system that is not contained in the ASP application root directory or its subdirectories. By default, **Enable parent paths** is set to **no**. This is the most secure setting and 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" on page 56. **Note:** The **Enable parent paths** setting does not add any restrictions to executing Java code. For example, if you want to restrict Java code to access files within the application directory, the proper permissions should be in the bean.policy file.

Stopping and Restarting the ASP Server (Admin Console)

There are times when you must stop and restart the ASP Server, such as when you're performing a product upgrade or putting configuration settings into effect. Use the following procedure to stop or restart the ASP Server. Stopping, starting, or restarting can take from several seconds to about one minute to execute. Restarting the ASP Server resets all **Session** and **Application** variables.

To stop, start, or restart the ASP Server

1. Open the Administration Console (see "Accessing the Administration Console" on page 18).
2. On the **Server Management** page, click **Stop**, **Start**, or **Restart**.



Note

While use of the Administration Console is strongly recommended, expert users can also stop and start the ASP Server from the command line. For more information, see "Stop/Start/Status ASP Server (Command Line)" on page 84.

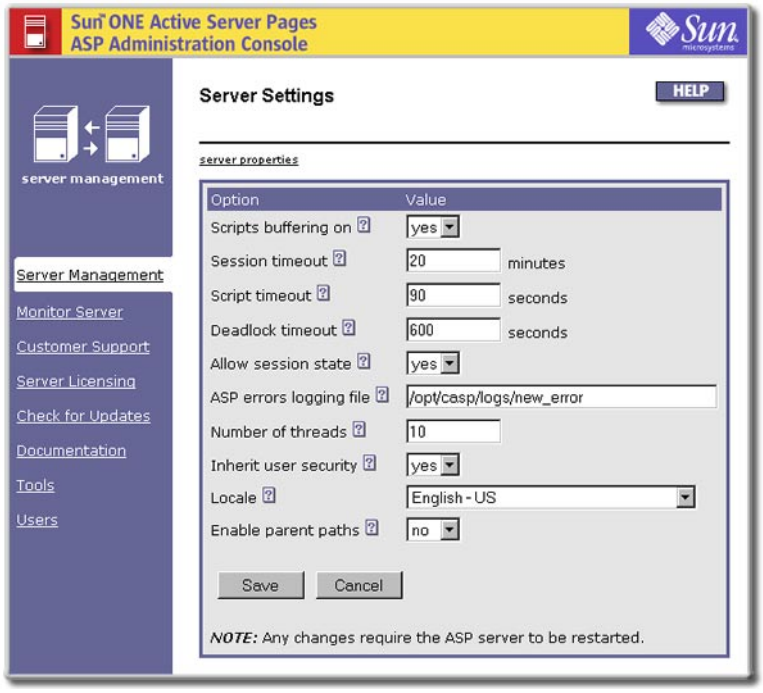
Enabling Session State

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, in order 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 (see "Accessing the Administration Console" on page 18).
2. On the **ASP Server** tab of the **Server Management** page, click **Settings**.

   The **Server Settings** page displays.



3. In the **Allow session state** drop-down list, select **yes** or **no**.
4. Click **Save** to save your changes, or **Cancel** to revert to the settings that were last 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" on page 261

Configuring International Support

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

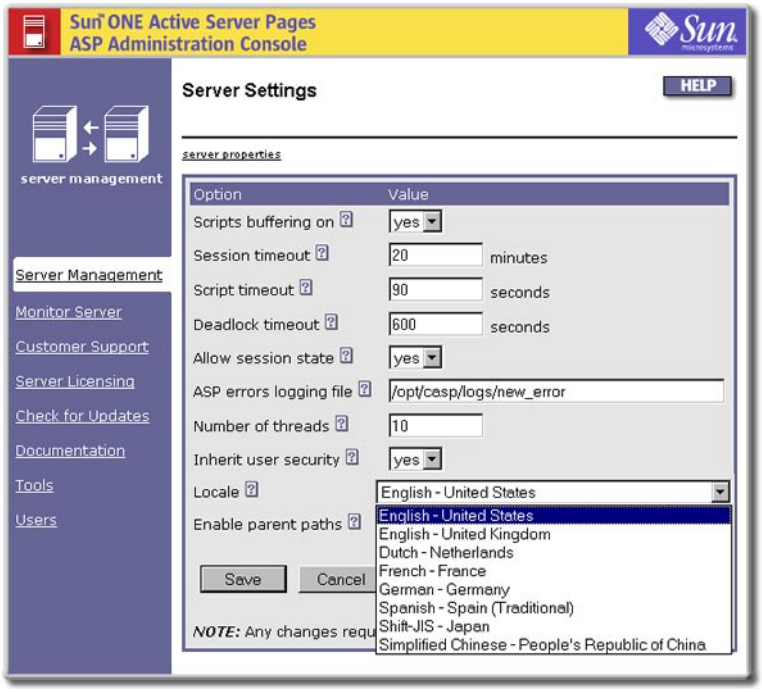
The following table lists supported languages, and the corresponding LCIDs (Local Language Identifiers) and code pages. The table is followed by the procedure that describes how to configure international support.

Language	LCID	Code page
English - US	1033	1252
English - British	2057	1252
Dutch	1043	1252
French	1036	1252
German	1031	1252
Japanese Shift-JIS	1041	932
Simplified Chinese	2052	936
Spanish	1034	1252

To configure international support

1. Open the Administration Console (see "Accessing the Administration Console" on page 18).
2. On the **ASP Server** tab of the **Server Management** page, click **Settings**.

   The **Server Settings** page displays.



3. In the **Locale** drop-down list, select the desired locale (the locales listed are the locales that are installed on the server). Click **Save** to save your changes, or **Cancel** to revert to the settings that were last saved.

   The **Server Management** page displays.

4. 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" on page 212

Creating Database Connections (ASP Server)

Sun ONE Active Server Pages enables you to 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 ONE ASP Server to connect to the database. Then the 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 information, see "Chapter 6, Configuring a Database" on page 103.

Sun ONE ASP provides a built-in ADO (ActiveX Data Object) 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 (Open Database Connectivity) drivers are called, and provides "containers" for storing the 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 detailed information about ADO, see "ADO Component Reference" on page 301.

For UNIX and Linux versions of Sun ONE ASP, the setup program automatically installs ODBC drivers for a number of different databases (ODBC drivers are not installed with Sun ONE ASP for Windows). The list of installed drivers can be viewed from the Sun ONE ASP Administration Console, as described in "Viewing the List of ODBC Drivers" on page 104. For Windows systems, the list of installed ODBC drivers can be viewed from the Windows Control Panel. See Microsoft documentation for more information.

Sun ONE ASP includes DataDirect SequeLink 5.3, which enables connections to remote computers running Microsoft Access or Microsoft SQL Server 6.5. For more information, see "Configuring SequeLink" on page 128.

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 ONE ASP, Web developers can specify the connection information for the database by using system DSNs (data source names), file DSNs, or DSN-less connection strings. The appropriate method depends on user preferences, and the environment in which Sun ONE ASP is running. For more information, see "Connecting to a Database" on page 197.

For enterprise applications, it is recommended that ASP developers use system DSNs. The system administrator can use the Sun ONE 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)" on page 105.

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

- 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 is 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 ONE ASP. An ODBC driver can bring down your ASP Server if incorrect parameters are

being passed. You should test your database connections on a nonproduction server.

The following example illustrates the relationship between Sun ONE 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
```

Defining ASP Applications (ASP Server)

Sun ONE Active Server Pages 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" on page 189. For information about accessing sample applications, which demonstrate the basics of building Sun ONE ASP applications, see "Developer Web Site" on page 15.

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

- Add the application from the Administration Console, as described in "Adding ASP Applications" on page 48.
- Enable ASP processing for a virtual host (referred to as virtual servers on Sun ONE Web Server), as described in "Enabling ASP for a Virtual Host" on page 54 and "Defining Applications in a Shared Environment" on page 74.
- Use the FrontPage Services file, as described in "Using the FrontPage Services File" on page 75.
- Add an application from within the Sun ONE ASP configuration file, or add an alias from within the Web server configuration files. These are advanced administration options for expert users, and are described in "Defining Applications on UNIX" on page 525.

Note

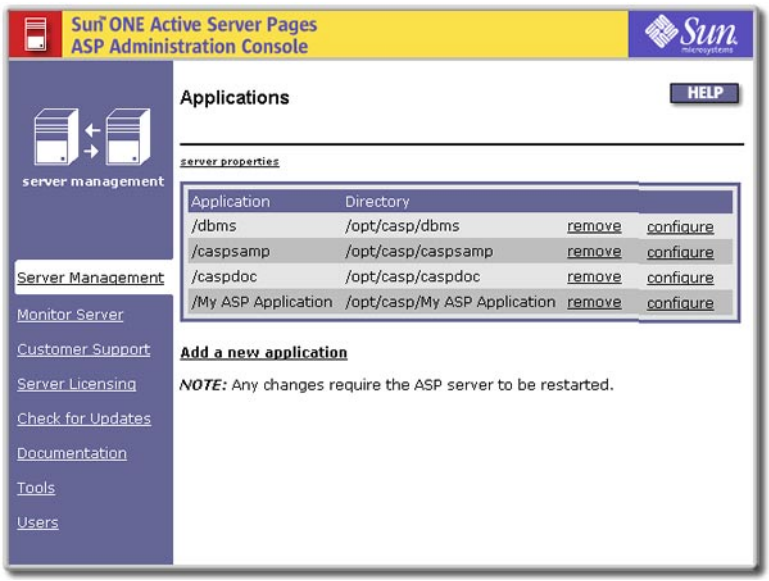
On Windows NT and Windows 2000, ASP applications are defined by adding aliases or virtual directories to the Web server. Sun ONE ASP treats each alias or virtual directory as an ASP application. With Sun ONE Web Server, ASP applications are defined by adding an "additional document directory" using the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

In this section:

Configuring ASP Applications

An ASP application must be defined on the ASP Server in order 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 ONE ASP Administration Console, as discussed in this section. However, if you need to configure an application in a hosted environment, see "Defining Applications in a Shared Environment" on page 74. For more information about defining FrontPage applications, see "Using the FrontPage Services File" on page 75.

You can define and configure an ASP application from the Administration Console **Applications** page, which displays when you click the **ASP Applications** link on the **Server Management** page.



The **Applications** page displays the list of ASP applications that are currently defined on the ASP Server, and provides access to settings for adding, removing, and configuring ASP applications:

- **Add a new application** creates a new application and associates it with the physical directory containing the global.asa file. See "Adding ASP Applications" on page 48.
- **remove** removes an ASP application from the ASP Server. See "Removing ASP Applications" on page 51.
- **configure** associates an ASP application with a physical directory containing the global.asa file. See "Editing ASP Application Settings" on page 52.

Note

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

See also:

"Defining ASP Applications (ASP Server)" on page 46

Adding ASP Applications

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 ONE Active Server Pages Administration Console. From the console, you add an application by giving the application a name, and by 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 (referred to as virtual servers on Sun ONE Web Server), do not use the following procedure. Instead, use the instructions in "Enabling ASP for a Virtual Host" on page 54. If the Web developers you support use FrontPage, see the description of FrontPage applications in "Using the FrontPage Services File" on page 75.

Note

On Windows NT and Windows 2000, ASP applications are defined by adding aliases or virtual directories to the Web server. Sun ONE Active Server Pages treats each alias or virtual directory as an ASP application. With Sun ONE Web Server, ASP applications are defined by adding an "additional document

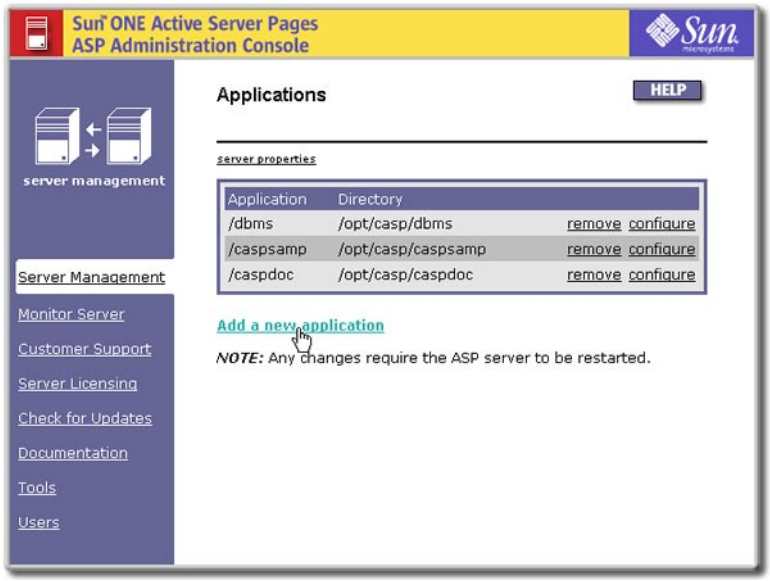
directory" using the Web server's Administration tool. With the Apache Web Server, ASP applications are defined by adding an alias to the httpd.conf file.

To define an ASP application that will not be served by a virtual host or virtual server, use the following procedure.

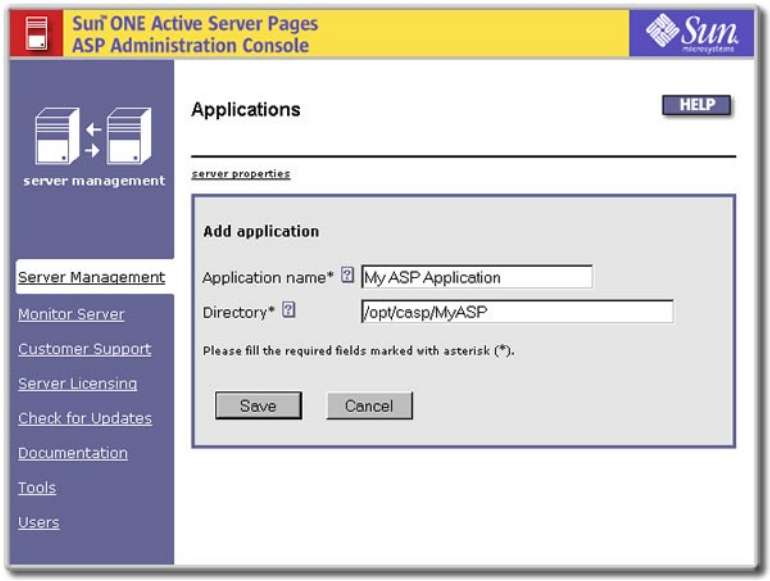
To add an ASP application

1. Open the Administration Console (see "Accessing the Administration Console" on page 18).
2. On the **ASP Server** tab of the **Server Management** page, 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.
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.
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 (ASP Server)" on page 46

"Configuring ASP Applications" on page 47

"Editing ASP Application Settings" on page 52

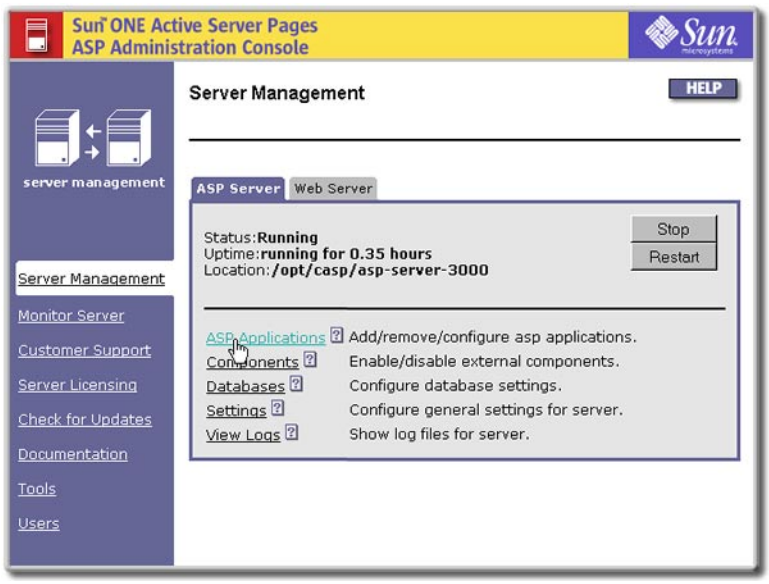
"Removing ASP Applications" on page 51

Removing ASP Applications

If you want the ASP Server to stop processing an ASP application, the ASP application must be removed from the ASP Server. 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" on page 47.

To remove an ASP application

1. Open the Administration Console (see "Accessing the Administration Console" on page 18).
2. On the **ASP Server** tab of the **Server Management** page, click **ASP Applications**.



   The **Applications** page displays, showing a list of currently configured ASP applications.
3. In line with the application you want to remove, click **remove**.
4. When prompted to confirm removal, 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 (ASP Server)" on page 46

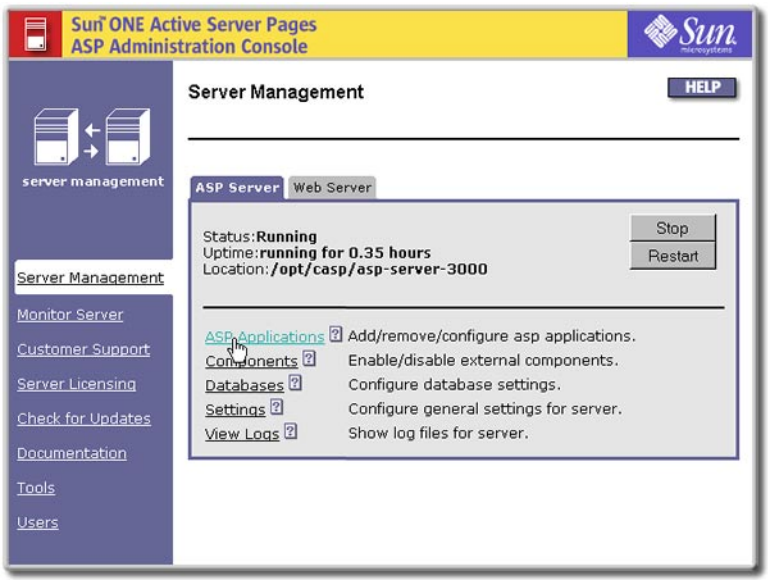
"Configuring ASP Applications" on page 47
"Editing ASP Application Settings" on page 52
"Adding ASP Applications" on page 48

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 ASP Applications" on page 48. 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" on page 47.

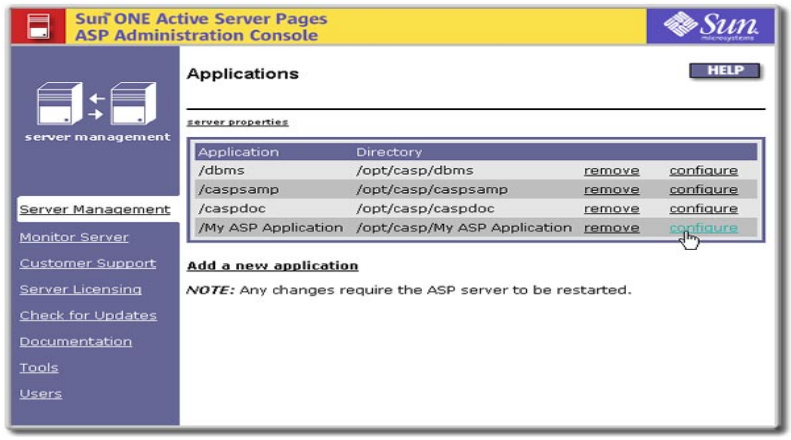
To edit ASP application settings

1. Open the Administration Console (see "Accessing the Administration Console" on page 18).
2. On the **ASP Server** tab of the **Server Management** page, click **ASP Applications**.

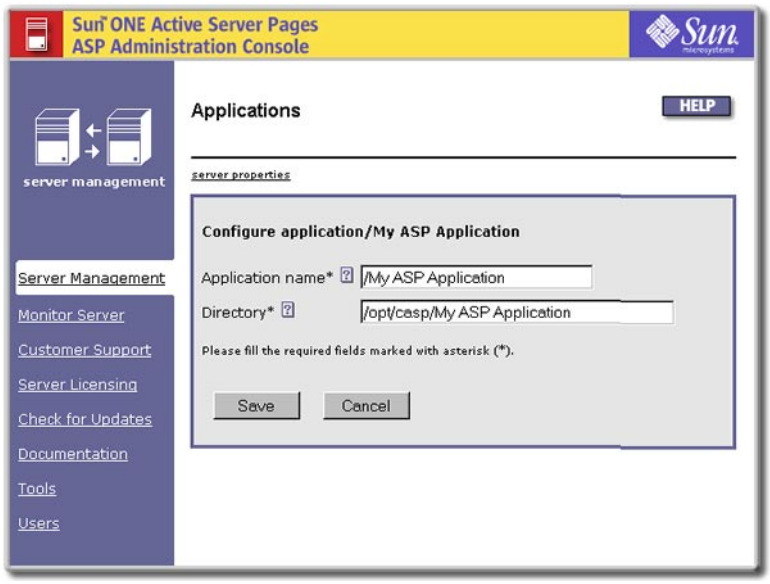


The **Applications** page displays, showing a list of currently configured ASP applications.

3. In line with the application 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, the 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 (ASP Server)" on page 46

"Configuring ASP Applications" on page 47

"Adding ASP Applications" on page 48

"Removing ASP Applications" on page 51

"Starting and Stopping the Web Server" on page 78

Enabling ASP for a Virtual Host

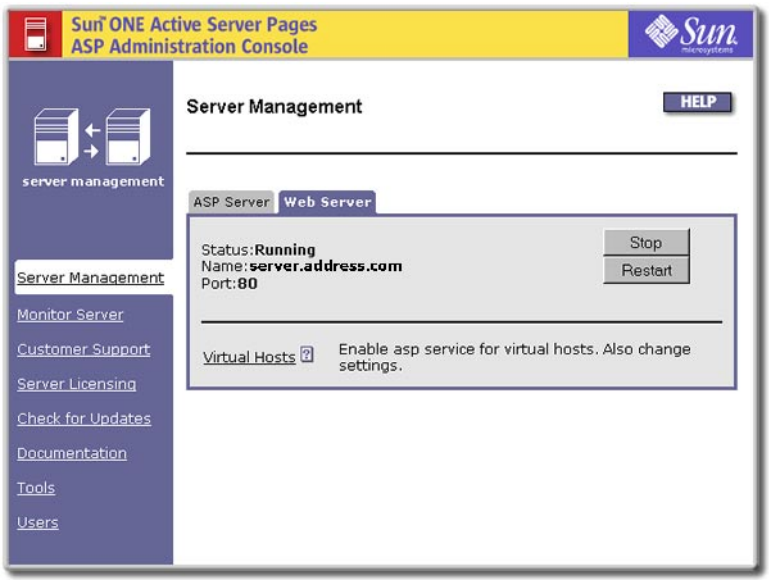
In a Web hosting environment that makes use of virtual hosts (referred to as virtual servers on Sun ONE Web Server), ASP applications are defined in a different manner than that described in "Adding ASP Applications" on page 48. This is because the ASP Server automatically recognizes ASP applications for each virtual host defined for the Web server.

This section describes how to use the Sun ONE 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" on page 47).

To enable or disable ASP processing for a virtual host

1. Open the Administration Console (see "Accessing the Administration Console" on page 18).
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 clear 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:

"Defining Applications in a Shared Environment" on page 74

"Add/Remove ASP in Virtual Hosts" on page 94 (CLI)

"Virtual Hosts" on page 581

Securing the Server

Certain Sun ONE Active Server Pages settings have important implications for the security of the ASP Server. This section addresses security issues for the server, and provides information to help ensure that settings are configured appropriately for your specific environment.

In this section:

"Configuring File System Access" on page 56

"Setting the Security Mode" on page 57

"Disabling Performance Monitoring" on page 60

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 ONE Active Server Pages 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** object 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 ONE ASP, with Microsoft ASP, when **Enable parent paths** is set to **no**, a text file can still be created outside of the application directory.)

When **Enable parent paths** is set to **yes**, the **FileSystemObject** object 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.

Caution

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 by making sure that that user has appropriately restricted file system permissions. For more information, see "Setting the Security Mode" on page 57.

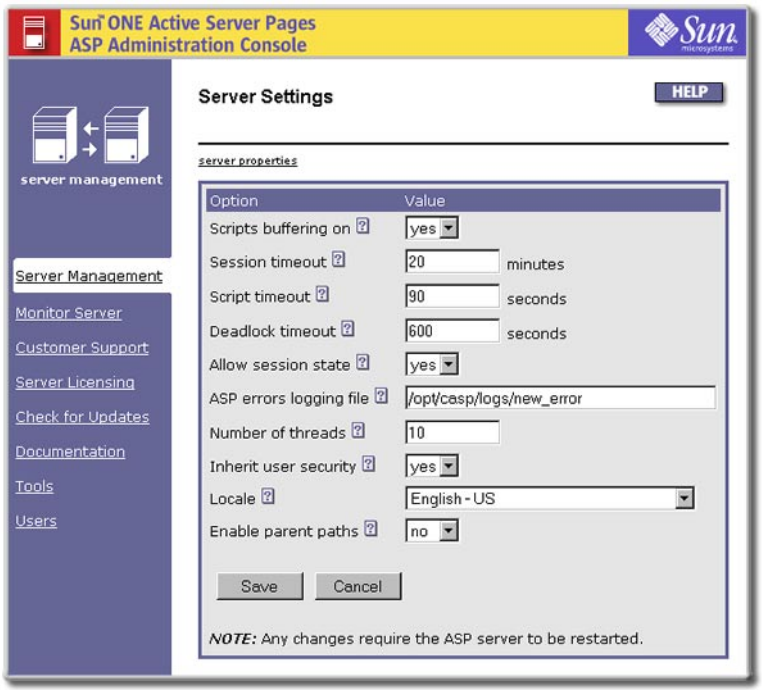
Note

The **Enable parent paths** setting does not add any restrictions to executing Java code. For example, if you want to restrict Java code to access files within the application directory, the proper permissions should be in the bean.policy file.

To configure file system access

1. Open the Administration Console (see "Accessing the Administration Console" on page 18).
2. On the **ASP Server** tab of the **Server Management** page, 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 (ASP Server)" on page 46

"Using Server-side Includes" on page 188

Setting the Security Mode

You can configure the Sun ONE ASP Server to run under Defined User Security mode or Inherit User Security mode (UNIX and Linux product versions). The appropriate mode depends on your Web hosting environment, and has important security implications for your server.

Caution

Be sure to read this section carefully, especially if you are running Sun ONE Web Server.

Inherit User Security Mode

Inherit User Security mode is available only for Sun ONE ASP running with the Apache Web server.

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 ONE ASP Administration Console, as described later in this section.

Sun ONE Web Server does not support Inherit User Security mode (the **Inherit user security** setting is not displayed in the Administration Console). To protect the security of your server when running Sun ONE ASP with Sun ONE Web Server, you should specify a user and group in the casp.cnfg file, as described in "Editing the Sun ONE ASP Configuration File" on page 517 (see the `[default machine]` keyword). The ASP Server then runs with the permissions of that user and group.

Defined User Security mode

Defined User Security mode mode is available for Sun ONE Active Server Pages running with both the Sun ONE and Apache Web servers, and 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 ONE 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 ONE ASP configuration files, such as casp.cnfg and odbc.ini. You enable this mode by setting **Inherit user security** to **no** in the Sun ONE ASP Administration Console (Apache) and then specifying a user and group in the casp.cnfg file (Apache and Sun ONE Web Server), as described in "Editing the Sun ONE ASP Configuration File" on page 517 (see the `[default machine]` section).

Caution

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.

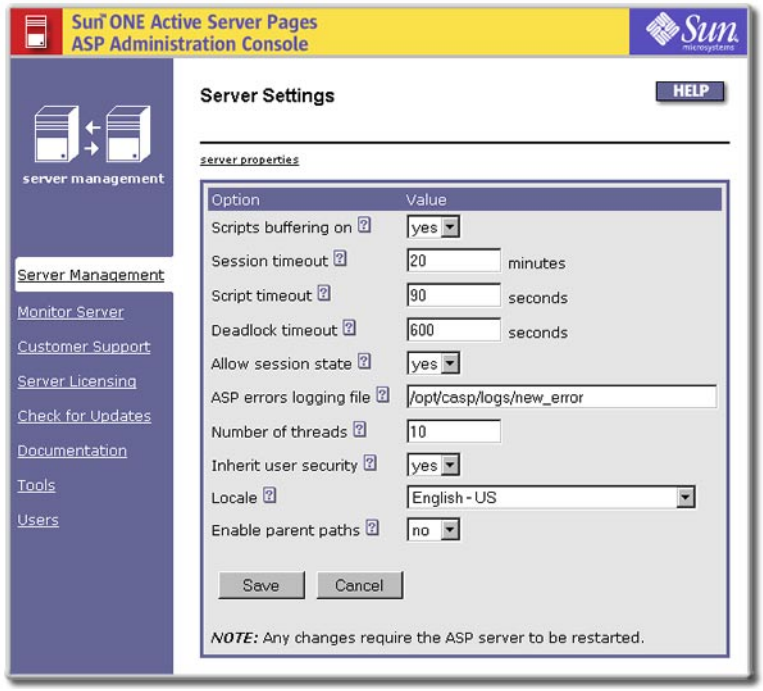
Note the following:

- Even if a user and 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.
- ADO logging will not be functional if **Inherit user security** is set to **yes**. For information about ADO logging, see "Enabling and Disabling ADO Logging" on page 133.
- The **Inherit user security** setting does not add any restrictions to executing Java code. For example, if you want to restrict Java code to access files within the application directory, the proper permissions should be in the bean.policy file.

To set the security mode

1. Open the Administration Console (see "Accessing the Administration Console" on page 18).
2. On the **ASP Server** tab of the **Server Management** page, 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.

   Caution

   If you select **no**, you should edit the casp.cnfg file to add a user and group for the ASP Server to run under, as described in "Editing the Sun ONE ASP Configuration File" on page 517. 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 **Cancel** to revert to the settings that were last 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" on page 56

Disabling Performance Monitoring

If you are running Sun ONE Active Server Pages 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.

Note

This feature is not available on Windows systems.

By default, Sun ONE ASP monitors server performance and displays this information on the Sun ONE ASP Administration Console **Server Monitoring** page, as described in "Monitoring ASP Server Performance" on page 61.

Sun ONE 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. Performance monitoring and the creation of these log files can be disabled by editing the `enablemonitoring` setting in the `[default machine]` section of the Sun ONE 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 the casp.cnfg file, see "Editing the Sun ONE ASP Configuration File" on page 517.

Viewing Information about the ASP Server

Sun ONE Active Server Pages provides several options for viewing information about the ASP Server. This section describes how to monitor real-time performance data, view diagnostic information, and log ASP errors.

In this section:

"Viewing the ASP Errors Log" on page 64

"Viewing Server Diagnostics" on page 65

See also:

"Optimizing ASP Server Performance" on page 66

"Diagnostic Applications" on page 13

Monitoring ASP Server Performance

Real-time information about ASP Server performance is displayed on the the **Server Monitoring** page of the Sun ONE ASP Administration Console.

If performance monitoring has been disabled as described in "Disabling Performance Monitoring" on page 60, you cannot view this server performance information. Disabling server performance monitoring is a recommended security precaution if you are running Sun ONE ASP in a shared Web hosting environment.

Note

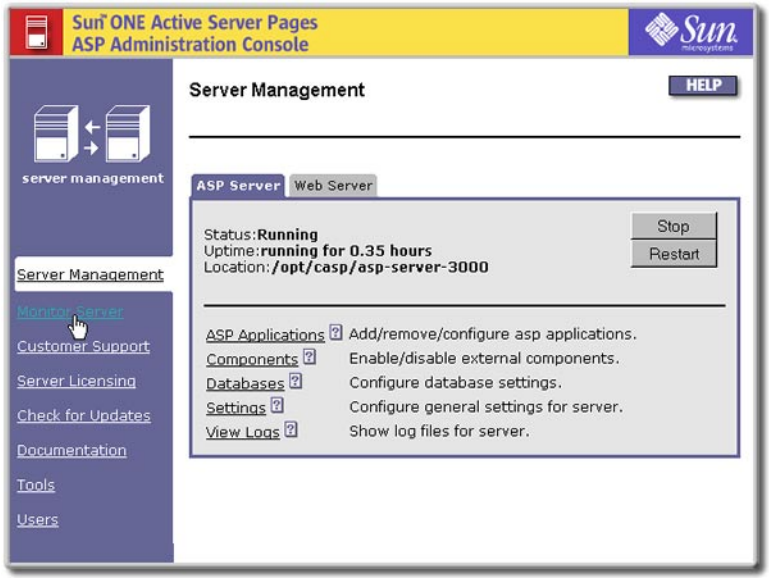
This feature is not available with Sun ONE Active Server Pages for Windows. On Windows systems, performance monitoring information is available via the Windows NT or Windows 2000 Performance Monitor. See Microsoft documentation for more information.

To view real-time information about the ASP Server

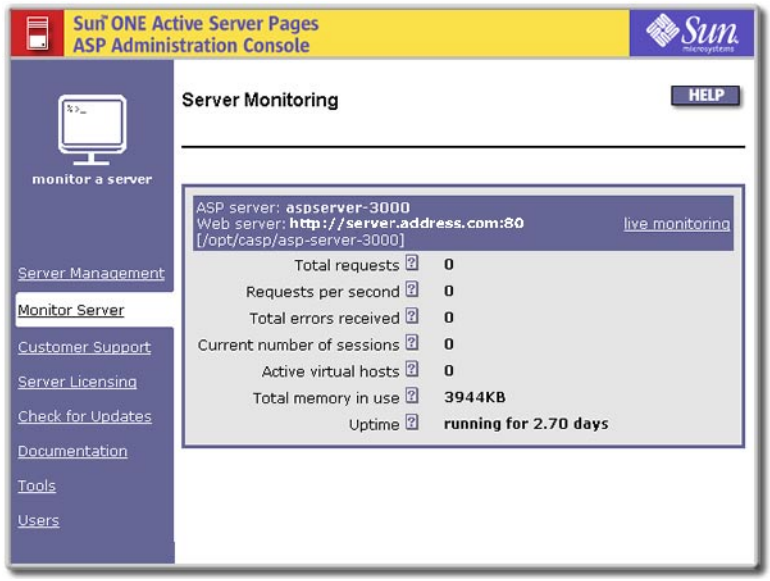
1. Open the Administration Console (see "Accessing the Administration Console" on page 18).

   The **Server Management** page displays.

2. In the left navigation pane, click **Monitor Server**.



The **Server Monitoring** page displays.



3. To continuously monitor the server, click **live monitoring**. This opens a separate window and displays information that is constantly updated.

The following table lists the information displayed on the **Server Monitoring** page.

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.
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" on page 37

"Viewing Information about the ASP Server" on page 60

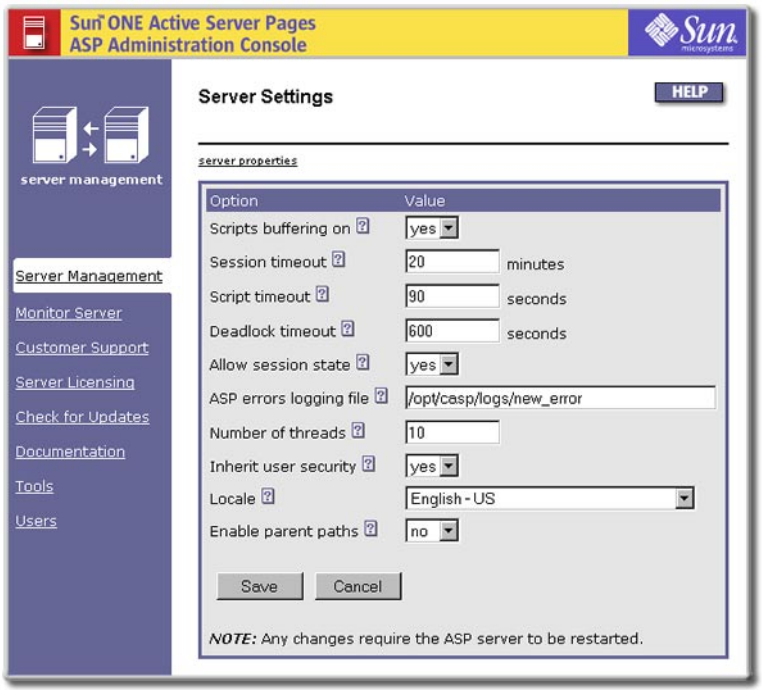
Enabling ASP Errors Logging

For Sun ONE Active Server Pages to log ASP errors, you must first enable logging. For information about viewing the log file, see "Viewing the ASP Errors Log" on page 64.

To enable ASP errors logging

1. Open the Administration Console (see "Accessing the Administration Console" on page 18).
2. On the **ASP Server** tab of the **Server Management** page, click **Settings**.

   The **Server Settings** page displays.



3. In the **ASP errors logging file** box, enter 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 **Cancel** to revert to the settings that were last saved.

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

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 ONE ASP installation directory (/opt/casp by default).

See also:

"Monitoring ASP Server Performance" on page 61

"Optimizing ASP Server Performance" on page 66

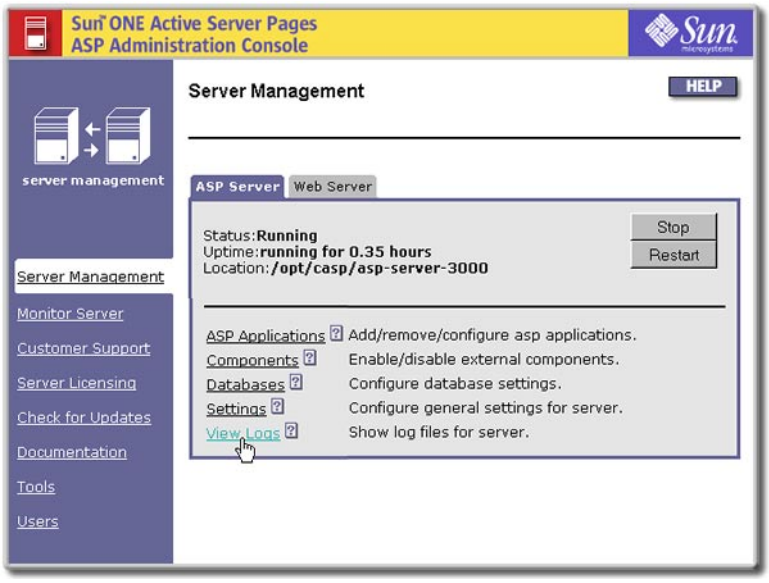
"Viewing Information about the ASP Server" on page 60

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 ONE Active Server Pages Administration Console. To log ASP errors and view the logging information, you must first enable logging as described in "Enabling ASP Errors Logging" on page 63.

To view the ASP errors log

1. Open the Administration Console (see "Accessing the Administration Console" on page 18).
2. On the **ASP Server** tab of the **Server Management** page, click **View Logs**.



The **Server Logs Files** page displays, showing the ASP errors that have been logged.

See also:

"Monitoring ASP Server Performance" on page 61

"Optimizing ASP Server Performance" on page 66

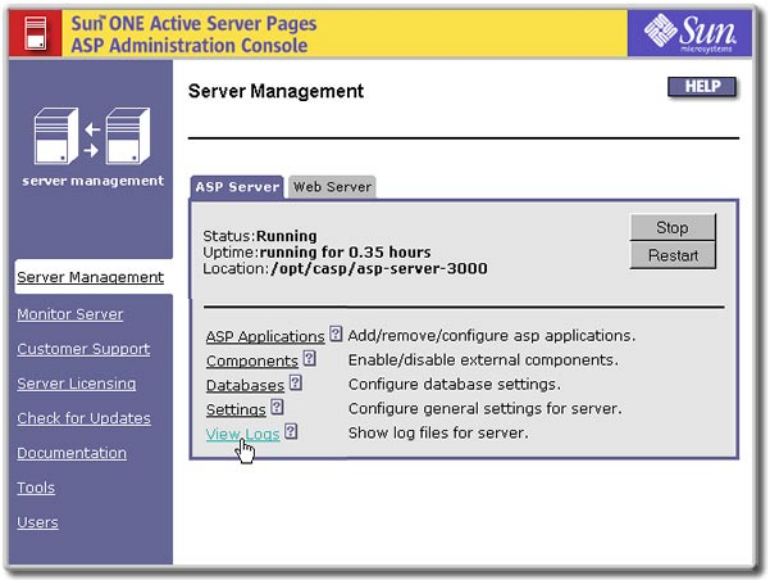
"Viewing Information about the ASP Server" on page 60

Viewing Server Diagnostics

You can view diagnostic information about the ASP Server from the **Server Logs Files** page of the Sun ONE Active Server Pages Administration Console, including when ASP engines were started and stopped, and what configuration changes have been made since Sun ONE ASP was installed.

To view server diagnostics

1. Open the Administration Console (see "Accessing the Administration Console" on page 18).
2. On the **ASP Server** tab of the **Server Management** page, 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" on page 63

"Monitoring ASP Server Performance" on page 61

"Optimizing ASP Server Performance" on page 66

"Viewing Information about the ASP Server" on page 60

"Diagnostic Applications" on page 13

Optimizing ASP Server Performance

Sun ONE Active Server Pages has many features that enhance its scalability and performance. This section describes those features.

In this section:

See also:

"Enabling ASP Errors Logging" on page 63

"Monitoring ASP Server Performance" on page 61

"Viewing Information about the ASP Server" on page 60

"Viewing the ASP Errors Log" on page 64

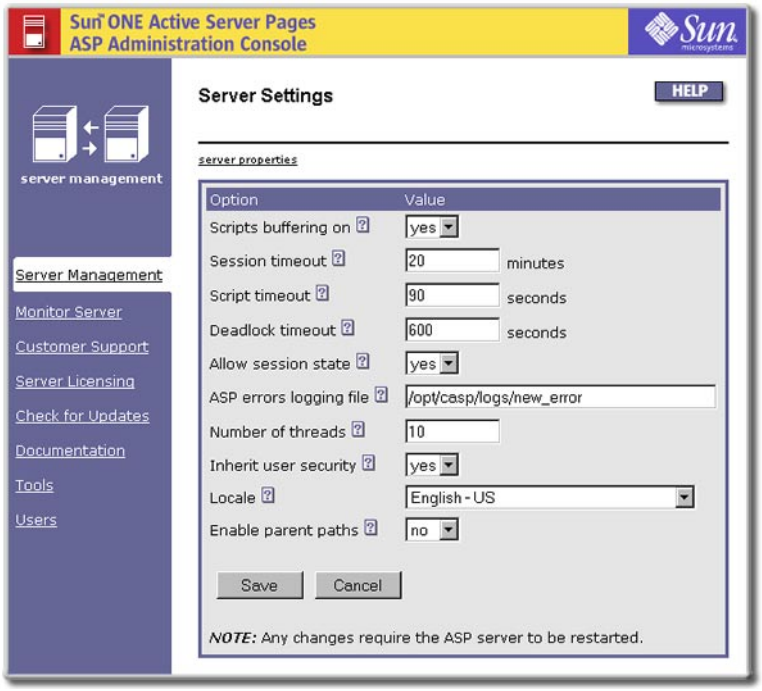
Enabling Scripts Buffering

Sun ONE Active Server Pages 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 to make it easier to debug problems with your ASP pages.

To enable or disable scripts buffering

1. Open the Administration Console (see "Accessing the Administration Console" on page 18).
2. On the **ASP Server** tab of the **Server Management** page, click **Settings**.

   The **Server Settings** page displays.



3. In the **Scripts buffering on** drop-down list, select **yes** or **no**.
4. Click **Save**, and then restart the ASP Server by clicking **Restart** on the **Server Management** page.

Note

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

Changing the Session Timeout Value

You can specify the number of minutes 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 resources for another session. The session timeout value is 20 minutes by default.

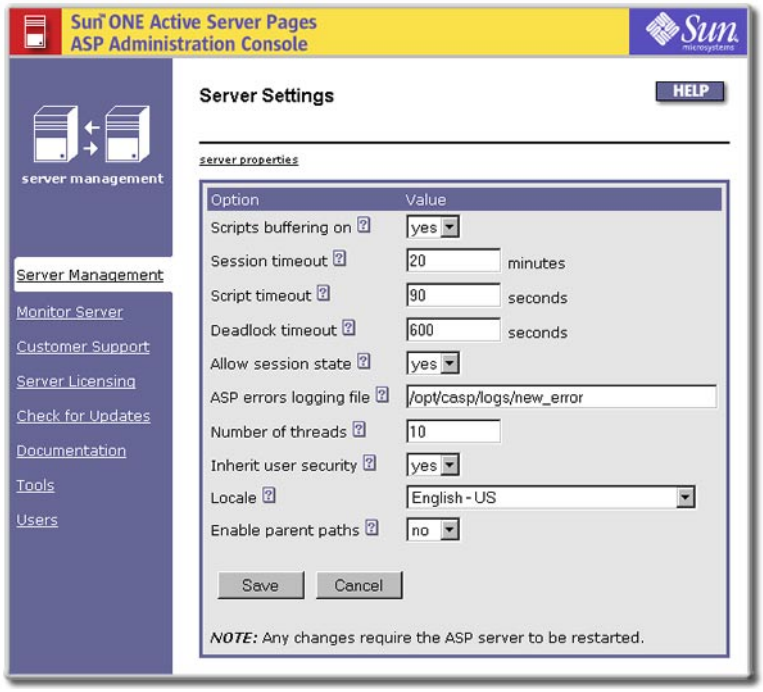
Note

A value specified for **SessionTimeout** in a script overrides this setting.

To change the session timeout value

1. Open the Administration Console (see "Accessing the Administration Console" on page 18).
2. On the **ASP Server** tab of the **Server Management** page, click **Settings**.

   The **Server Settings** page displays.



3. In the **Session timeout** box, specify 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** on the **Server Management** page.

Note

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

Changing the Script Timeout Value

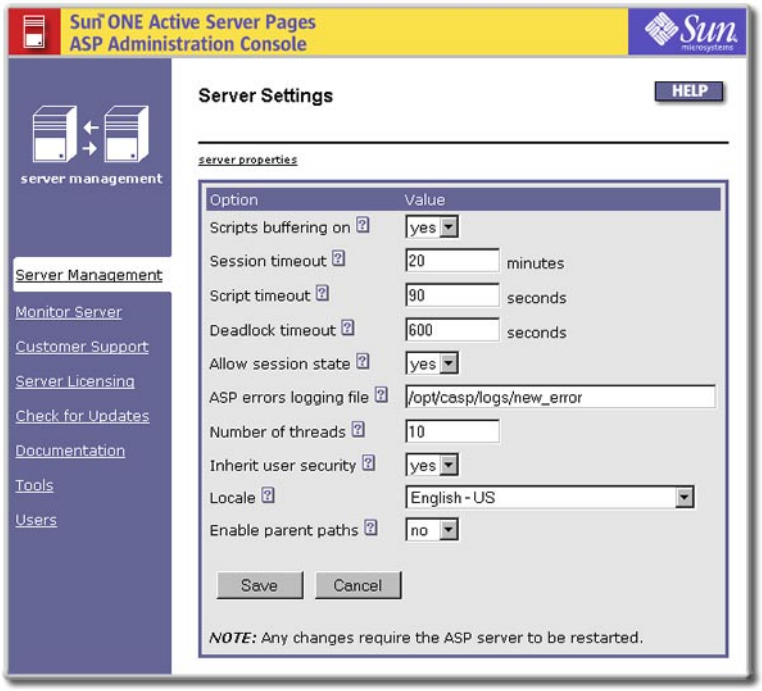
You can specify the number of seconds the ASP Server waits for an ASP page to finish processing before canceling the page request. Setting a script timeout prevents a malfunctioning ASP page from indefinitely engaging server resources. Enabling the ASP Server to cancel a page request frees up resources for another session. The script timeout value is 90 seconds by default.

If the deadlock timeout is set to a value lower than the script timeout, the ASP engine will restart once the time specified for the deadlock timeout has elapsed. For more information about the deadlock timeout and deadlock recovery, see "Configuring Engine Deadlock Recovery" on page 69.

To change the script timeout value

1. Open the Administration Console (see "Accessing the Administration Console" on page 18).
2. On the **ASP Server** tab of the **Server Management** page, click **Settings**.

   The **Server Settings** page displays.



3. In the **Script timeout** box, specify the number of seconds after which a script should time out.

   If the deadlock timeout is set to a value lower than the script timeout, the ASP engine will restart after the time specified for the deadlock timeout has elapsed.
4. Click **Save**, and then restart the ASP Server by clicking **Restart**.

Note

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

Configuring Engine Deadlock Recovery

You can configure the amount of time that should elapse before the ASP Server is considered deadlocked and the engine is restarted. This functionality helps to alleviate potential problems caused if an ASP engine becomes completely deadlocked and stops servicing requests, a condition that could result from failed processes, thread contention, locked database connections, and so on. The deadlock timeout value is set to 600 seconds (10 minutes) by default.

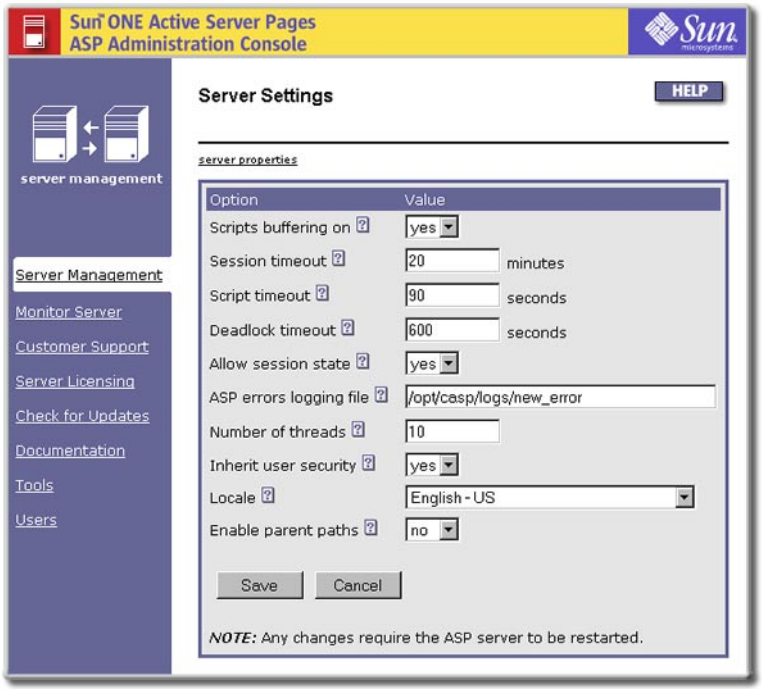
Note of the following:

- If the deadlock timeout is set to a value lower than the script timeout, the ASP engine will restart once the time specified for the deadlock timeout has elapsed. For more information about the script timeout value, see "Changing the Script Timeout Value" on page 68.
- The default of 600 seconds may or may not be the best setting for you. Selecting the "correct" deadlock timeout value is not an exact science and depends on your specific circumstances. Web sites that pass around large amounts of data from databases will require a different timeout value than those with pages that merely manipulate a few user-specified strings.
- Be careful about setting the timeout value too high. A deadlock condition exists until the deadlock timeout has elapsed and the ASP Server is restarted, so your Web site could potentially be down for the length of time specified for the deadlock timeout.

To change the deadlock timeout value

1. Open the Administration Console (see "Accessing the Administration Console" on page 18).
2. On the **ASP Server** tab of the **Server Management** page, click **Settings**.

   The **Server Settings** page displays.



3. In the **Deadlock timeout** box, specify the number of seconds that should elapse before the ASP Server is considered deadlocked and the engine is restarted.

   If the deadlock timeout is set to a value lower than the script timeout, the ASP engine will restart after the time specified for the deadlock timeout has elapsed.

4. Click **Save**, and then restart the ASP Server by clicking **Restart**.

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

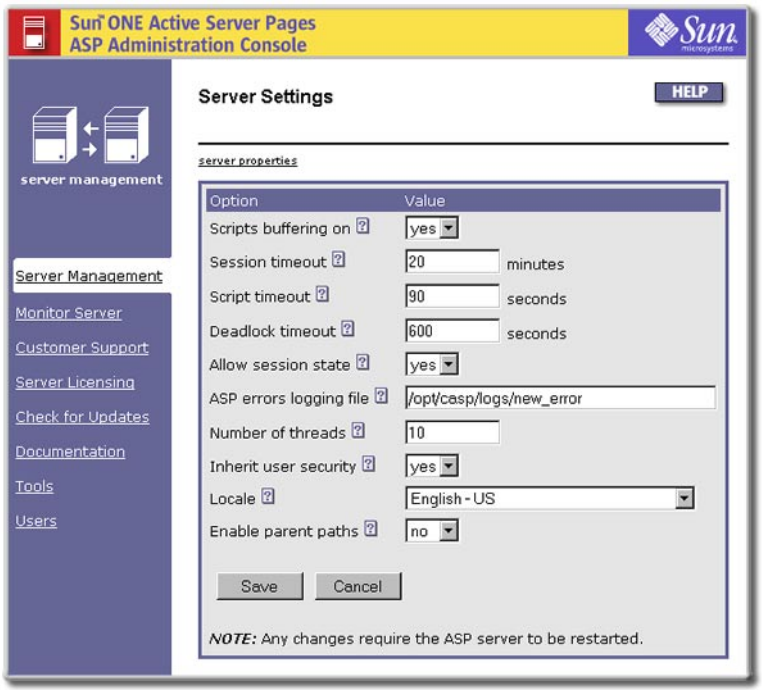
Configuring Multi-threading

By default, the number of threads handled simultaneously by the Sun ONE ASP Server is 5 for both Solaris and Linux. 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 up to 20 threads is recommended. DO NOT set this to a number greater than 20.

To configure multi-threading

1. Open the Administration Console (see "Accessing the Administration Console" on page 18).
2. On the **ASP Server** tab of the **Server Management** page, click **Settings**.

   The **Server Settings** page displays.



3. In the **Number of threads** box, enter the maximum number of threads you want to have running at once. The default is 10 for Solaris, and 5 for Linux.

4. Click **Save**, and then click **Yes** when the **Server Management** page displays to restart the Web server and ASP Server.

Note

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

Precompiling ASP Pages

The Sun ONE 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 processed more quickly 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 ONE ASP Server to share open database connections among multiple users 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" on page 131.

Load Balancing

Sun ONE Active Server Pages 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 ONE 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.

Shared Web Hosting Environments

Sun ONE Active Server Pages supports the scenario in which users share physical hardware and a Web server, such as with an Internet Service Provider. In a shared Web hosting environment, a single Web server installation answers requests for multiple domain names by using virtual hosts (referred to as virtual servers on Sun ONE Web Server).

This section provides information about running Sun ONE ASP in a shared Web hosting environment.

In this section:

See also:

"Securing the Server" on page 55

"Chapter C, Advanced Administration Options" on page 515

Creating Database Connections in a Shared Environment

With Sun ONE Active Server Pages, ASP developers can specify the connection information for a database by using either system DSNs (data source names), file DSNs, or DSN-less connection strings. The appropriate method depends on user preferences and the environment in which Sun ONE 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 ONE 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)" on page 105.

However, in a shared Web hosting environment, such as with an Internet Service Provider, system DSNs pose 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 is more appropriate in a Web hosting environment.

See also:

"Chapter 6, Configuring a Database" on page 103

"Creating Database Connections (ASP Server)" on page 44

"Using DSN-less Connection Strings" on page 200

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, an ASP application is defined by "adding" it to the ASP Server, as described in "Configuring ASP Applications" on page 47.

However, in a shared Web hosting environment in which virtual hosts are being used (referred to as virtual servers on Sun ONE Web Server), such as with an Internet Service Provider, applications are not "added" 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 may be some situations, however, in which you want to enable or disable ASP processing for a particular virtual host. You can do this as described in "Enabling ASP for a Virtual Host" on page 54. FrontPage users can also define an application as described in "Using the FrontPage Services File" on page 75.

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 ASP Applications" on page 48), 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 ONE 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:

1. In the `[applications]` section of the Sun ONE 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 Sun ONE ASP Configuration File" on page 517.

2. 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:

- 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.
- If the User Configuration file appears in the document root of a virtual host, the ASP applications are applied only to that virtual host, and not to others.

See also:

"Configuring ASP Applications" on page 47

Using the FrontPage Services File

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 ONE Active Server Pages automatically looks for the services.cnf file in the /_vti_pvt subdirectory, and treats the entries it finds in this file as ASP applications. Applications defined in the services.cnf file are dynamically recognized by Sun ONE ASP, and do not require the ASP Server to be restarted. Sun ONE 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:

- The files in the application must be located within the document root directory of the Web server (or virtual host).
- 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" on page 47.

See also:

"Defining Applications in a Shared Environment" on page 74

4 Managing the Web Server

Sun ONE Active Server Pages 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, but some settings can be accessed from the Sun ONE ASP Administration Console.

This chapter describes how to manage the Web server from the Administration Console.

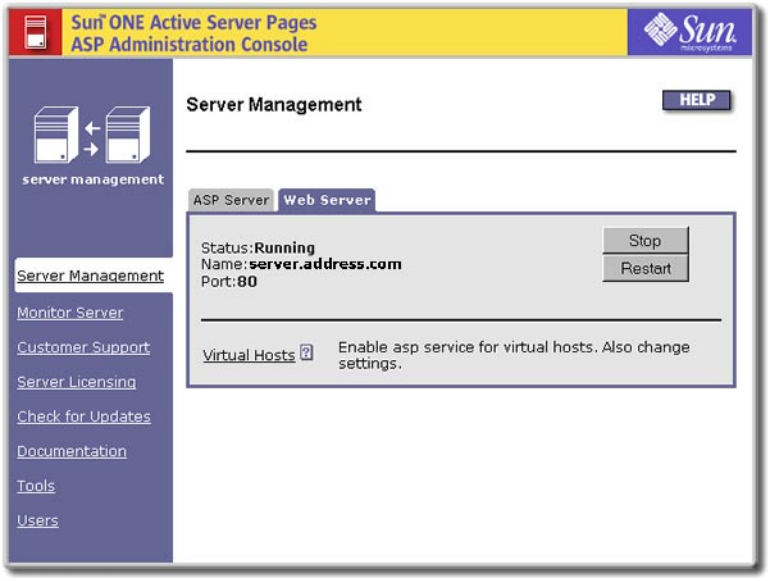
In this chapter:

See also:

"Chapter 3, Managing the ASP Server" on page 35

Server Management Overview (Web)

Certain settings for the Web server can be managed from the **Web Server** tab on the **Server Management** page in the Sun ONE ASP Administration Console.



On this tab you can view information about the Web server, start and stop the Web server, and enable ASP processing for individual virtual hosts (referred to as virtual servers on Sun ONE Web Server).

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" on page 78.
- The **Virtual Hosts** link displays an option for enabling and disabling ASP processing for individual virtual hosts. For more information, see "Enabling ASP for a Virtual Host" on page 54 and "Defining Applications in a Shared Environment" on page 74.

See also:

"Configuring the Web Server after Installation" on page 79

"Changing the Linkage" on page 87

"Server Management Overview (ASP)" on page 36

Starting and Stopping the Web Server

The Sun ONE Active Server Pages Administration Console can be used to start, stop, and restart the Web server with which the Sun ONE ASP Server is configured to run. You can also view the status of the Web server (whether it's stopped or running).

To start, stop, and restart the Web Server

1. Open the Administration Console (see "Accessing the Administration Console" on page 18).
2. On the **Server Management** page, click the **Web Server** tab.

   The **Web Server** tab displays.
3. Click **Start**, **Stop,** or **Restart**.

See also:

"Chapter 4, Managing the Web Server" on page 77

Configuring the Web Server after Installation

If you chose not to configure a Web server to run with Sun ONE Active Server Pages during installation, you will be prompted to install one the first time you open the Sun ONE ASP Administration Console.

To configure a Web server after installation

1. Open the Administration Console (see "Accessing the Administration Console" on page 18).

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

2. Perform the desired action:

   - Select the option button of the Web server you want to configure, and then click **OK**.

     - 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 click **OK**.

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.

After completing the previous steps, the Administration Console **Server Management** page appears. From this page, you can configure the ASP Server and Web server, as described in "Chapter 3, Managing the ASP Server" on page 35 and "Chapter 4, Managing the Web Server" on page 77 (and other related sections).

Note

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

See also:

"Changing the Linkage" on page 87

Changes to Web Server Configuration Files

When a Web server is configured to run with Sun ONE Active Server Pages (either during installation or after), the setup program makes certain changes to Web server configuration files. These changes are described in this section.

In this section:

Changes to Sun ONE Web Server Configuration Files

When Sun ONE Active Server Pages is installed on a computer running Sun ONE Web Server (formerly iPlanet Web Server, Enterprise Edition), the following changes are made to the Web server configuration files:

- Lines are added to the beginning of the obj.conf file (for Sun ONE Web Server 4.1) or the magnus.conf file (for Sun ONE 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_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/sunos5_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_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/linux2_optimized/netscape_6.x/nes_casp
  2.so"
  Init fn="caspinit" casplib="[C-ASP_INSTALL_DIR]/asp-[server]-
  ```

```
[PORT]"
```

- The following lines are added 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>
```

- The following lines are added 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
```

- Support for ASAP WebShow is commented out 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 ONE Active Server Pages supports only one instance of **LoadObjects** in the Sun ONE Web Server magnus.conf file.

Changes to Apache Configuration Files

When Sun ONE Active Server Pages is installed on a computer running Apache Web Server, the following changes are made to the Web server configuration file (httpd.conf):

- These lines are added:

```
AddHandler chiliasp .asp
AddHandler chiliasp .asa
CaspLib [C-ASP_INSTALL_DIR]/asp-[server]-[PORT]
```

- These lines are added:

  On Solaris:

```
LoadModule  casp2_module
[C-ASP_INSTALL_DIR]/module/sunos5_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
```

- This line is added:

  ```
  MaxRequestsPerChild 30000
  ```

Enabling FrontPage Publishing

FrontPage Server Extensions are a set of server-side applications (CGI programs) that enable you 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 IIS (Internet Information Server). To enable this capability, FrontPage Server Extensions must be installed, and FrontPage authoring must be enabled on the Web server.

Sun ONE ASP supports but does not install FrontPage Server Extensions. You must obtain them from Microsoft, and can do so at:

http://msdn.microsoft.com/library/en-us/dnservext/html/fpse02unix.asp

Once the extensions are installed, you must take additional steps to enable users to publish their pages to the server. For information about using FrontPage in a shared environment, see "Using the FrontPage Services File" on page 75.

Note

While Sun ONE ASP enables you to run ASP pages generated by 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 **Enable parent paths** is set to **no** by default. 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 **Enable parent paths** to **yes**, or 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" on page 56.

See also:

"Using the FrontPage Services File" on page 75

5 Command-line Management

Most settings for Sun ONE Active Server Pages should be configured from the Administration Console, a browser-based GUI used for managing Sun ONE ASP. However, while it is strongly recommended that the Administration Console be used for product configuration, expert users do have the option to perform certain tasks from the command line.

Some command-line functionality is provided in both an interactive and a non-interactive mode. As the names imply, the interactive mode requires users to respond to prompts, while the non-interactive mode requires no interaction beyond issuing commands that specify path information, file locations, and so on. Functionality for both modes is provided by a singleton master script called `configure-server`, which serves as the primary entry point for managing the ASP Server. The `configure-server` script is located in the Sun ONE ASP installation directory (/opt/casp by default).

This chapter describes some but not all of the command-line functionality provided in this release of Sun ONE Active Server Pages. Complete command-line documentation is available, however, and can be accessed as described in "Command-line Help" on page 84.

Caution

This functionality is for expert users of Sun ONE Active Server Pages. Take great care when making the changes described in this chapter. Changes you make could require a complete reinstall of Sun ONE ASP and could void your eligibility for customer support. You should back up your data before making any changes.

In this chapter:

Command-line Help

Not all command-line functionality provided in this release of Sun ONE Active Server Pages is documented in this guide. Complete command-line documentation can be accessed from the Sun ONE ASP installation directory (/opt/casp by default) by typing the following at the command prompt:

```
./configure-server -help
```

The Help page provides complete documentation for command-line functionality, indicating what each function does and any necessary parameters. If you are performing command-line management, it is strongly recommended that you consult this resource.

Using configure-server

The `configure-server` script serves as the primary entry point for command-line management of the ASP Server. The `configure-server` script is located in the Sun ONE Active Server Pages installation directory (/opt/casp by default).

To use configure-server

From the Sun ONE ASP installation directory, type the following at the command prompt:

```
./configure-server
```

and then the desired command(s), as specified in the remainder of this chapter.

Note

If you make any changes to casp.cnfg, you must restart the Sun ONE ASP Server. If you make any changes to the `[default application]` section in casp.cnfg, you must restart both the Sun ONE ASP Server and the Web server.

Stop/Start/Status ASP Server (Command Line)

This functionality is used to stop, start, and query the status of the ASP Server.

Note

For information about performing these tasks from the Sun ONE Administration Console, see "Stopping and Restarting the ASP Server (Admin Console)" on page 41.

Stop/Start/Status: configure-server

This section describes the use of the `configure-server` script to stop, start, and query the status of the ASP Server. The scriptable interface is described in the following table.

Function	Explanation
function=enginectl	Allows you to stop and start a specific ASP engine, and to retrieve its status.
Parameters	
engine=<*path/name*> (user specified)	If a path is specified, the full path information will be used to identify the ASP engine being modified. If a name is specified (such as asp-server-3000), the path is assumed to reside within the current installation.
restart	Restarts the associated engine. Has an exit code of 0 if the operation succeeds, non-0 if the operation fails.
start	Starts the associated engine. Has an exit code of 0 if the operation succeeds, non-0 if the operation fails.
stop	Stops the associated engine. Has an exit code of 0 if the operation succeeds, non-0 if the operation fails.
status	Returns the status of the associated engine. Has an exit code of 0 if the ASP server is running, non-0 if it isn't.

Example Usage

```
cd <install dir>
./configure-server function=enginectl engine=asp-server-3000 restart


if ./configure-server function=enginectl engine=asp-server-3000 start;
then
  echo "The server successfully started."
else
  echo "The server failed to start."
fi
```

Stop/Start/Status: caspctrl

Some tasks can also be performed using the caspctrl script. This section describes how to use the script, and lists available options.

To use the caspctrl script

From the Sun ONE Active Server Pages installation directory (in /opt/casp/asp-server-*xxxx*), type the following at the command prompt:

```
./caspctrl
```

followed by the desired option(s). The correct format is as follows:

```
./caspctrl (-v|version) (-vc|verbose) [-]
(startdaemon|stopdaemon|starteng|stopeng|startall|stopall|viewlog|
clearlog|status)
```

The following table lists the options provided by caspctrl.

Option	Explanation
version	Reports the version of Sun ONE ASP.
verbose	Enables the Sun ONE ASP engine to output status messages to stdout.
startdaemon	Starts the Sun ONE ASP Server daemon.
stopdaemon	Stops the Sun ONE ASP Server daemon (plus any running ASP engines).
starteng	Starts Sun ONE ASP engine(s) on all Sun ONE ASP computers in the configuration with running daemons.
stopeng	Stops the Sun ONE ASP engine(s) on all computers running Sun ONE ASP in the configuration with running daemons.
startall	Starts the Sun ONE ASP daemon and engine(s) on single-computer installations of Sun ONE ASP.
viewlog	Enables you to view the Sun ONE ASP Server log.
clearlog	Clears the Sun ONE ASP Server log.
status	Reports the status of each Sun ONE ASP Server in the configuration.

Add/Delete/Reconfigure ASP Servers

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

This section describes how to change the ASP Server-to-Web Server linkage from the command line (both interactive and noninteractive), and includes information about:

- Adding an ASP engine to an installation
- Deleting an ASP engine from an installation
- Reconfiguring an ASP engine within an installation

See also:

"Starting on System Boot" on page 90

Changing the Linkage

Use the following procedure to change the ASP Server-to-Web Server linkage that was specified during the installation of Sun ONE Active Server Pages. Settings will not be migrated.

Note

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

To change the ASP Server-to-Web server linkage

1. From the Sun ONE ASP installation directory (/opt/casp by default), type the following at the command prompt:

   ```
   ./configure-server
   ```

2. At the prompt, select **Configure Sun ONE ASP**.
3. Select **Change the Web server-to-Sun ONE ASP 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 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 ONE ASP.

- or -

- Choose **2. Custom configuration** if you are an experienced user of Sun ONE 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 ONE 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 ONE 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 (choose another Web server).

Add ASP Server

This functionality is used to add an ASP engine to an installation. The scriptable interface (`configure-server`) is described in the following table.

Function	Explanation
function=add-server	Creates a new Sun ONE ASP-to-Web server linkage.
Parameters	
webserver_conf=<*file path*> (user specified)	Specifies the location of the Web server configuration file, used to create the ASP Server-to-Web server linkage.
webserver_binary=<*file path*> (default=auto-detect)	Specifies the location of the Web server binary file.
webserver_restart=<*yes*|*no*> (default=yes)	Specifies whether to restart the Web server once Sun ONE ASP has been configured.
webserver_overwrite=<*yes*|*no*> (default=no)	Specifies whether to overwrite any previous Sun ONE ASP installation currently linked to the specified Web server.
install_docs=<*yes*|*no*> (default=yes)	Specifies whether Sun ONE ASP documentation should be made visible to the Web server (/caspdoc).

Function	Explanation
start_asp=<*yes*|*no*> (default=yes)	Specifies whether Sun ONE ASP should be started once configuration is complete.
start_asp_onboot=<*yes*|*no*>	Specifies whether Sun ONE ASP should be started on system reboot. For interactive command-line functionality for this setting, see "Starting on System Boot" on page 90.

Example Usage

```
cd <install dir>
./configure-server function=add-server
    webserver_conf=/etc/httpd/conf/httpd.conf

./configure-server function=add-server
    webserver_conf=/home/deanb/my-httpd.conf
    webserver_binary=/usr/bin/httpd start_asp_onboot=no
```

Delete ASP Server

This functionality is used to delete an ASP engine from an installation. The scriptable interface (`configure-server`) is described in the following table.

Function	Explanation
function=delete-server	Deletes an existing Sun ONE ASP-to-Web server linkage.
Parameters	
engine=<*path*|*name*> (user specified)	If a path is specified, the full path information will be used to identify the ASP engine being modified. If a name is specified (such as asp-server-3000), the path is assumed to reside within the current installation.

Example Usage

```
cd <install dir>
./configure-server function=delete-server engine=asp-server-3000
```

Reconfigure ASP Server

This functionality is used to reconfigure an ASP engine within the installation. The scriptable interface (`configure-server`) is described in the following table.

Function	Explanation
function=change-server	Creates a new Sun ONE ASP-to-Web server linkage.
Parameters	
engine=<*path*|*name*> (user specified)	If a path is specified, the full path information will be used to identify the ASP engine being modified. If a name is specified (such as asp-server-3000), the path is assumed to reside within the current installation.
webserver_conf=<*file path*> (user specified)	Specifies the location of the Web server configuration file to which you want to reconfigure the specified ASP engine.
webserver_binary=<*file path*> (default=auto-detect)	Specifies the location of the Web server binary file.
webserver_restart=<*yes*|*no*> (default=yes)	Specifies whether to restart the Web server once Sun ONE ASP has been configured.

Example Usage

```
cd <install dir>
./configure-server function=change-server engine=asp-server-3000
  webserver_conf=/etc/httpd/conf/httpd.conf

./configure-server function=change-server engine=asp-server-3000
  webserver_conf=/etc/httpd/conf/httpd.conf
  webserver_binary=/usr/bin/httpd
```

Starting on System Boot

The option to automatically start the Sun ONE ASP Server on system boot is configured during installation (the default is to start on system boot). To enable or disable the "start on system boot" functionality, use the following procedure.

Note

For a related command-line option, see "Add ASP Server" on page 88.

To start Sun ONE ASP on system boot

1. From the Sun ONE ASP installation directory (/opt/casp by default), type the following at the command prompt:

   ```
   ./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, the Sun ONE ASP Server will start on system boot.

Changing casp.cnfg Settings

The noninteractive functionality described in this section is used to list, modify, create, and delete settings in casp.cnfg, the Sun ONE Active Server Pages configuration file. This functionality is useful primarily for Internet Service Providers and other users who need to make changes to several ASP engines quickly and simultaneously.

For most users, modifications to casp.cnfg should be made using the Sun ONE ASP Administration Console. For more information about using the Administration Console to manage the ASP Server, see "Chapter 3, Managing the ASP Server" on page 35. Expert users of Sun ONE ASP can also make manual changes to casp.cnfg, as described in "Editing the Sun ONE ASP Configuration File" on page 517.

Note

If you make any changes to casp.cnfg, you must restart the Sun ONE ASP Server. If you make any changes to the `[default application]` section in casp.cnfg, you must restart both the Sun ONE ASP Server and the Web server.

The scriptable interface (`configure-server`) is described in the following table.

Function	Explanation
function=configure-engine	Allows you to change casp.cnfg settings.
Parameters	
engine [0-9]*= <*path*|*name*> (user specified)	If a path is specified, the full path information will be used to identify the ASP engine being modified. If a name is specified (such as asp-server-3000), the path is assumed to reside within the current installation.
section (user specified)	This is the section within casp.cnfg (such as section=default machine) that contains the key you want to add, delete, modify, or view.
key (user specified)	This is the key in the specified section that you want to add, delete, modify, or view.
value (user specified)	This is the value for the key.

Function	Explanation
mode=<*add*|*del*|*view*> (default=add)	Determines which action(s) to take with the provided parameters: - The "add" mode adds the specified key/value under the specified section. If that key is already associated with a value, it will be overwritten. - The "del" mode removes any entry with the specified key under the specified section. - The "view" mode prints the current contents of the specified key to stdout, in the following format: <casp.cnfg>: <section>: <key>: <value> **Caution:** Using mode=del could prevent the ASP engine from starting. It is strongly recommended that mode=del be used sparingly, if at all. Appropriate uses are listed in "Deleting casp.cnfg Settings" on page 94.
override=<*yes*|*no*>	This parameter specifies whether error checking should be provided when modifying a section/key value. This checking disallows creating new keys or modifying or viewing keys that are not user configurable. **Caution:** It is strongly recommended that you do not use override=yes. Doing so could render your server inoperable.

See also:

"Editing the Sun ONE ASP Configuration File" on page 517

"Add/Remove ASP in Virtual Hosts" on page 94

"Add/Remove Applications" on page 96

Examples: Listing casp.cnfg Settings

The following examples demonstrate listing casp.cnfg settings.

Example 1

```
./configure-server function=configure-engine engine=asp-server-3209
  mode=view 'section=default application' key=bufferingon
```

Result

```
/opt/casp.johnd6/asp-server-3209/casp.cnfg: default application:
 bufferingon: yes
```

Example 2

```
./configure-server function=configure-engine engine0=asp-server-3209
```

```
 engine1=asp-server-3224 mode=view
 'section=default application' key=bufferingon
```

Result

```
/opt/casp.johnd6/asp-server-3209/casp.cnfg: default application:
 bufferingon: yes
/opt/casp.johnd6/asp-server-3224/casp.cnfg: default application:
 bufferingon: yes
```

Examples: Changing casp.cnfg Settings

The following examples demonstrate changing casp.cnfg settings.

Example 1

```
./configure-server function=configure-engine engine=asp-server-3209
  mode=add 'section=default application' key=bufferingon value=no
./configure-server function=configure-engine engine=asp-server-3224
  mode=add 'section=default application' key=bufferingon value=no
./configure-server function=configure-engine engine0=asp-server-3209
  engine1=asp-server-3224 mode=view 'section=default application'
  key=bufferingon
```

Result

```
/opt/casp.deanb4/asp-server-3209/casp.cnfg: default application:
 bufferingon: no
/opt/casp.deanb4/asp-server-3224/casp.cnfg: default application:
 bufferingon: no
```

Example 2

```
./configure-server function=configure-engine engine0=asp-server-3209
  engine1=asp-server-3224 mode=add 'section=default application'
  key=bufferingon value=yes
./configure-server function=configure-engine engine0=asp-server-3209
  engine1=asp-server-3224 mode=view 'section=default application'
  key=bufferingon
```

Result

```
/opt/casp.janed6/asp-server-3209/casp.cnfg: default application:
 bufferingon: yes
/opt/casp.janed6/asp-server-3224/casp.cnfg: default application:
 bufferingon: yes
```

Deleting casp.cnfg Settings

Caution

Using mode=del could prevent the ASP engine from starting. It is strongly recommended that mode=del be used sparingly, if at all.

Using mode=del is really only appropriate for such actions as removing virtual hosts or applications from the casp.cnfg file, because those actions can be considered both common and relatively safe.

Add/Remove ASP in Virtual Hosts

The noninteractive functionality described in this section is used to enable and disable ASP in virtual hosts. The scriptable interface (`configure-server`) is described in the following table.

Note

To enable or disable ASP processing for a virtual host using the Sun ONE ASP Administration Console, see "Enabling ASP for a Virtual Host" on page 54. Expert users of Sun ONE ASP can also make manual changes to casp.cnfg, as described in "Editing the Sun ONE ASP Configuration File" on page 517.

Function	Explanation
function=configure-engine	Allows you to change casp.cnfg settings.
Parameters	
engine [0-9]*= <*path*|*name*> (user specified)	If a path is specified, the full path information will be used to identify the ASP engine being modified. If a name is specified (such as asp-server-3000), the path is assumed to reside within the current installation.
section (user specified)	This is the section within casp.cnfg (such as section=default machine) that contains the key you want to add, delete, modify, or view.
key (user specified)	This is the key in the specified section that you want to add, delete, modify, or view.

Function	Explanation
value (user specified)	This is the value for the key.
mode=<*add*|*del*|*view*> (default=add)	Determines which action(s) to take with the provided parameters: - The "add" mode adds the specified key/value under the specified section. If that key is already associated with a value, it will be overwritten. - The "del" mode removes any entry with the specified key under the specified section. - The "view" mode prints the current contents of the specified key to stdout, in the following format: <casp.cnfg>: <section>: <key>: <value> **Caution:** Using mode=del could prevent the ASP engine from starting. It is strongly recommended that mode=del be used sparingly, if at all. Appropriate uses are listed in "Deleting casp.cnfg Settings" on page 94.
override=<*yes*|*no*>	This parameter specifies whether error checking should be provided when modifying a section/key value. This checking disallows creating new keys or modifying keys that are not user configurable. **Caution:** It is strongly recommended that you do not use override=yes. Doing so could render your server inoperable.

Example: Adding Virtual Hosts

The following example demonstrates adding a virtual host.

Example

```
./configure-server function=configure-engine engine=asp-server-3209
  mode=add 'section=virtual hosts' key='allow_all' value='no'
./configure-server function=configure-engine engine=asp-server-3209
  mode=add 'section=virtual hosts' key='www.foobar.com'
```

Result

Your ASP engine will be accessible only on virtual host www.foobar.com.

Examples: Removing Virtual Hosts

The following examples demonstrate removing virtual hosts.

Example 1

```
./configure-server function=configure-engine engine=asp-server-3209
  mode=add 'section=virtual hosts' key='allow_all' value='yes'
```

Result

Your ASP engine will be accessible to any virtual host.

Example 2

```
./configure-server function=configure-engine engine=asp-server-3209
  mode=del 'section=virtual hosts' key='www.foobar.com'
```

Result

This removes the "www.foobar.com" entry from the casp.cnfg file, as it no longer has any affect on the server.

Add/Remove Applications

This functionality is used to add and remove applications. The scriptable interface (`configure-server`) is described in the following table.

Note

For information about adding and remove applications using the Sun ONE ASP Administration Console, see "Defining ASP Applications (ASP Server)" on page 46.

Function	Explanation
function=configure-engine	Allows you to change casp.cnfg settings.
Parameters	
engine [0-9]*= <*path*|*name*> (user specified)	If a path is specified, the full path information will be used to identify the ASP engine being modified. If a name is specified (such as asp-server-3000), the path is assumed to reside within the current installation.

Function	Explanation
section (user specified)	This is the section within casp.cnfg (such as section=default machine) that contains the key you want to add, delete, modify, or view.
key (user specified)	This is the key in the specified section that you want to add, delete, modify, or view.
value (user specified)	This is the value for the key.
mode=<*add*|*del*|*view*> (default=add)	Determines which action(s) to take with the provided parameters: - The "add" mode adds the specified key/value under the specified section. If that key is already associated with a value, it will be overwritten. - The "del" mode removes any entry with the specified key under the specified section. - The "view" mode prints the current contents of the specified key to stdout, in the following format: <casp.cnfg>: <section>: <key>: <value> **Caution:** Using mode=del could prevent the ASP engine from starting. It is strongly recommended that mode=del be used sparingly, if at all. Appropriate uses are listed in "Deleting casp.cnfg Settings" on page 94.
override=<*yes*|*no*>	This parameter specifies whether error checking should be provided when modifying a section/key value. This checking disallows creating new keys or modifying keys that are not user configurable. **Caution:** It is strongly recommended that you do not use override=yes. Doing so could render your server inoperable.

Examples: Adding Applications

The following examples demonstrate adding applications.

Example 1

```
./configure-server function=configure-engine engine=asp-server-3209
  mode=add 'section=applications' key=/tmp value=/tmp
```

Result

Your ASP engine will associate the /tmp virtual URI with the /tmp local directory.

Example 2

```
./configure-server function=configure-engine engine=asp-server-3209
  mode=add 'section=applications' key=/foobar value=/home/me/foo/bar
```

Result

Your ASP engine will associate the virtual URI /foobar with the local path /home/me/foo/bar.

Examples: Removing Applications

The following examples demonstrate removing applications.

Examples

```
./configure-server function=configure-engine engine=asp-server-3209
  mode=del 'section=applications' key=/tmp

./configure-server function=configure-engine engine=asp-server-3209
  mode=del 'section=applications' key=/foobar
```

List/View/Add/Edit/Delete ODBC DSNs

This functionality can be used to do the following:

- Display a list of supported database types and the corresponding DSN attributes.
- Display a list of all DSN names that have been configured (excluding template DSNs).
- Show keys and values for all attributes in the DSN.
- Add, edit, and delete DSNs.

Note

For information about configuring DSNs using the Sun ONE ASP Administration Console, see "Chapter 6, Configuring a Database" on page 103.

Show Database Types

This functionality is used to display a list of supported database types and the corresponding DSN attributes. The scriptable interface (`configure-server`) is described in the following table.

Function	Explanation
function=show_dbinfo (no parameters)	Displays a list of supported database types, and for each type, the list of configurable DSN attributes. Has an exit code of 0 if the operation succeeds, non-0 if the operation fails.

Example Usage

```
./configure-server function=show_dbinfo
```

List all DSNs

This functionality is used to display a list of all DSN names that have been configured (excluding template DSNs). The scriptable interface (`configure-server`) is described in the following table.

Function	Explanation
function=list_dsns (no parameters)	Displays a list of all configured DSN names. Does not include template DSNs.

Example Usage

```
./configure-server function=list_dsns
```

View Specific DSNs

This functionality is used to show keys and values for all attributes in the DSN. The scriptable interface (`configure-server`) is described in the following table.

Function	Explanation
function=view_dsn	Shows keys and values for all attributes in the DSN, including both configurable and the default.
Parameter	
name=<*DSN name*> (user specified)	Name of the DSN as displayed in list_dsns.

Example Usage

```
./configure-server function=view_dsn name=mysql_test
```

Add/Edit DSNs

This functionality is used to create a new DSN or modify an existing one. The scriptable interface (`configure-server`) is described in the following table.

Function	Explanation
function=new_dsn	Allows you to create a new DSN or modify an existing one. Has an exit code of 0 if the operation succeeds, non-0 if the operation fails.
Parameters	
name=<*new DSN name*> (user specified)	This is the name for the new DSN. If a DSN with that name already exists, it will be replaced.
type=<*database type*> (user specified)	This must be one of the database names displayed by show_dbinfo. If the <value> contains spaces, the <key>=<value> parameter must be enclosed in spaces. The numeric index displayed next to the database name by show_dbinfo may also be passed as the value.
Optional Parameters	One or more parameters of the form <key>=<value>, where <key> is one of the allowable attributes for the database type displayed by show_dbinfo.

Example Usage

```
./configure-server function=new_dsn name=mysql_test type=MySQL
Server=test Database=mysql Port=3306 User=root Password=root
```

Delete DSNs

This functionality is used to delete a DSN. The scriptable interface (`configure-server`) is described in the following table.

Function	Explanation
function=delete_dsn	Deletes a previously configured DSN.
Parameter	
name=<*DSN name*> (user specified)	Name of the DSN as displayed in list_dsns.

Example Usage

```
./configure-server function=delete_dsn name=mysql_test
```

Uninstalling Sun ONE ASP

On UNIX and Linux systems, Sun ONE Active Server Pages is uninstalled by running the script named `uninstall`, which is located in the Sun ONE ASP installation directory (/opt/casp by default).

When you run the uninstall program, you can delete all directories and files contained in the Sun ONE ASP installation directory. Before running 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 ONE ASP.

To uninstall Sun ONE ASP

1. From the Sun ONE Active Server Pages installation directory (/opt/casp by default), type the following at the command prompt:

   ```
   ./uninstall
   ```

2. Choose the number of the desired uninstall method:

   - Choose **1. Uninstall the entire product** to remove all Sun ONE 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 ONE ASP association, responding to the prompts as desired.
5. Delete the directories and files in the Sun ONE ASP installation directory.

6 Configuring a Database

Sun ONE Active Server Pages 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" on page 197.)

For UNIX and Linux versions of Sun ONE ASP, the setup program automatically installs the ODBC drivers for a number of databases (ODBC drivers are not installed with Sun ONE ASP for Windows). You can view the list of installed drivers from the Sun ONE ASP Administration Console, as described in "Viewing the List of ODBC Drivers" on page 104 (customer support is provided only for ODBC drivers that are installed with Sun ONE ASP). For some types of databases, however, 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 (see "Configuring SequeLink" on page 128). In addition, the system administrator might want to create system DSNs to make it easier for developers to connect with databases.

This chapter describes how to create and edit DSNs, and how to configure the ASP Server to connect with supported databases.

In this chapter:

See also:

"Creating Database Connections (ASP Server)" on page 44

Viewing the List of ODBC Drivers

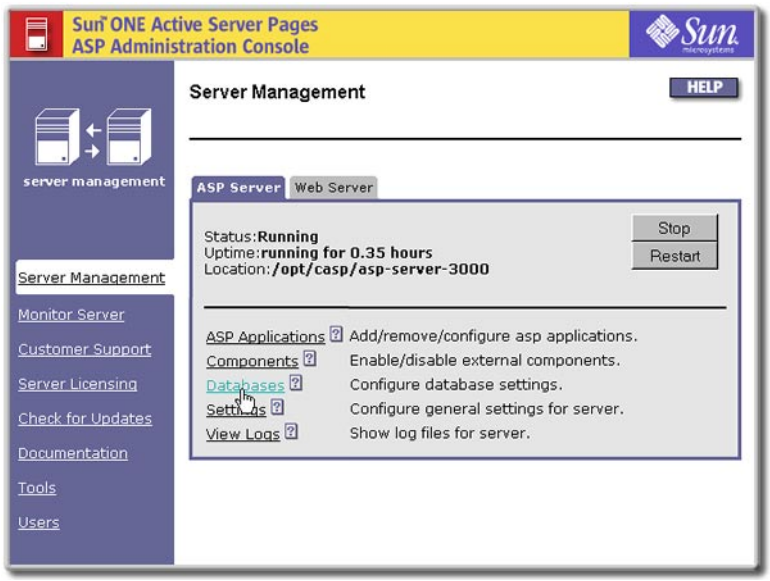
Sun ONE Active Server Pages enables you to connect to a variety of ODBC-compliant databases by using the appropriate ODBC driver. To verify whether Sun ONE ASP supports a specific version of a database, you can view the list of ODBC drivers included with Sun ONE ASP on the **Drivers** tab of the Sun ONE ASP Administration Console **Databases** page.

Note

Sun ONE Active Server Pages for UNIX and Linux installs the ODBC drivers for a number of databases. Drivers are not installed with Sun ONE ASP for Windows. For Windows systems, the list of installed ODBC drivers can be viewed from the Windows Control Panel. See Microsoft documentation for more information.

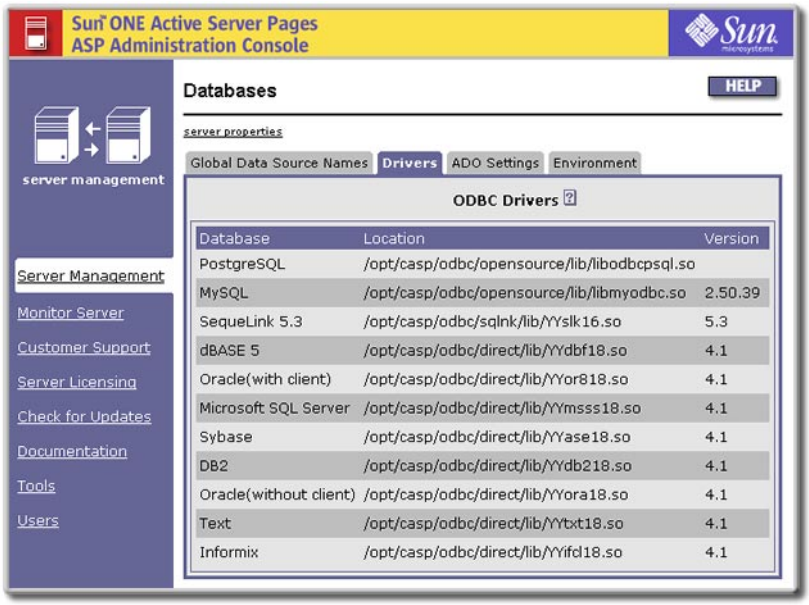
To view the list of ODBC Drivers

1. Open the Administration Console (see "Accessing the Administration Console" on page 18).
2. On the **ASP Server** tab of the **Server Management** page, click **Databases**.



3. On the **Databases** page, click the **Drivers** tab.

   The **ODBC Drivers** page displays, showing the list of installed ODBC drivers, and their locations in the file system.



See also:

"Supported in This Release" on page 5

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 DSN (data source name) 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 **Global Data Source Names** tab of the Sun ONE 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.

Please note the following:

- Expert users can also perform certain DSN-related tasks from the command line. For more information, see "Chapter 5, Command-line Management" on page 83.
- 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, rather than system DSNs. For a discussion of these security issues, see "Creating Database Connections (ASP Server)" on page 44.

- For Windows systems, DSNs are created and managed from the Windows Control Panel. See Microsoft documentation for more information.

This section describes how to add, remove, edit, and test system DSNs.

In this section:

Adding a DSN

Use the following procedure to add a system DSN to the ASP Server. When a DSN is added, Sun ONE Active Server Pages automatically sets the correct parameters for the databases. You can edit these parameters if necessary, as described in "Editing a DSN" on page 110.

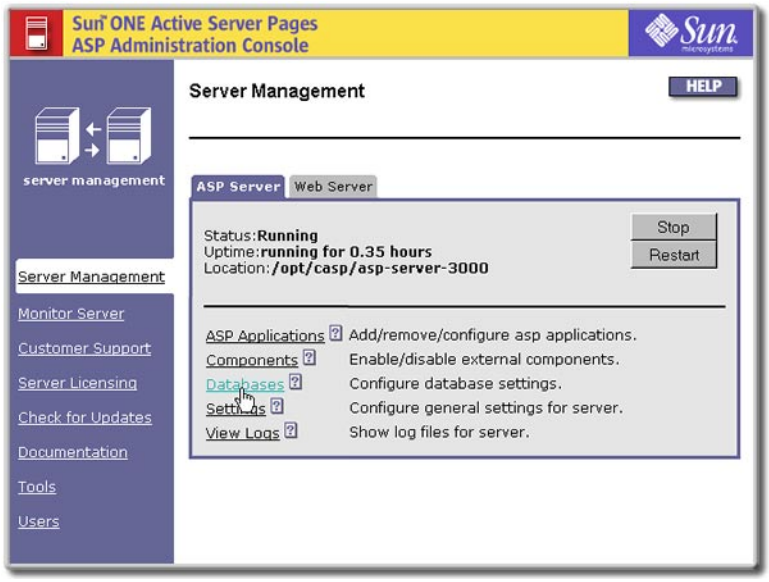
Note

For Microsoft Access and Microsoft SQL Server 6.5, Sun ONE ASP includes an ODBC driver for Microsoft SQL Server 7.0 and 2000, but you must use SequeLink 5.3 for connecting to Microsoft Access and Microsoft SQL Server 6.5 databases. You can create a DSN for SequeLink using the following procedure for adding a system DSN.

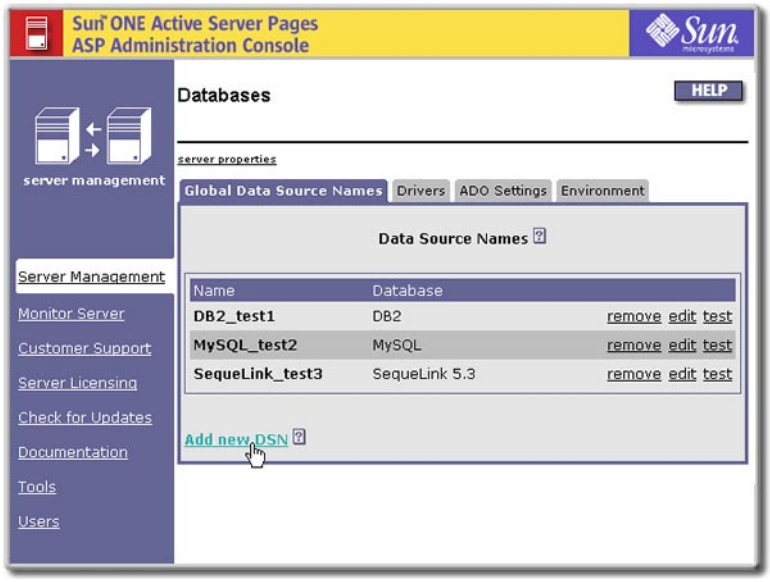
For Oracle and Informix (with client), after adding a DSN, you must also define database environment variables, as described in "Configuring the Database Environment" on page 112. Unless the environment variables have been previously set, after you finish adding a new DSN, the Sun ONE ASP Administration Console displays the appropriate page on which to configure these settings.

To add a system DSN

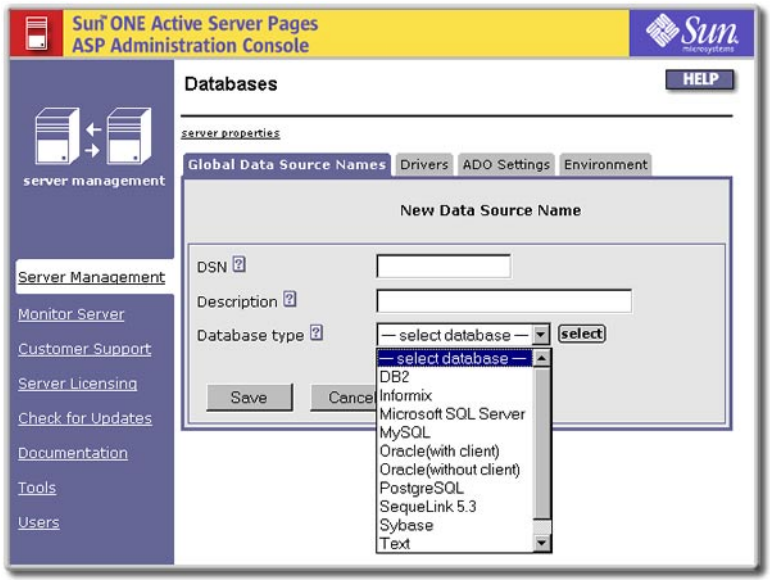
1. Open the Administration Console (see "Accessing the Administration Console" on page 18).
2. On the **ASP Server** tab of the **Server Management** page, click **Databases**.



3. On the **Databases** page, click **Add new DSN** in the bottom left of the screen.



The **New Data Source Name** page displays.



4. In the **DSN** box, type a name for the DSN.
5. 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 5.3**).
7. In the remaining text boxes, provide the requested information (if necessary, see the information specific to your database in "Configuring Database Parameters" on page 115).
8. To save your changes, click **Save**, and then click **Done**.

   - or -

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

The new DSN displays in the **Data Source Names** list. After adding a DSN, you should also test it, as described in "Testing a DSN" on page 111.

Note

For Windows systems, data source names are created and managed from the Windows Control Panel. See Microsoft documentation for more information.

See also:

"Configuring Data Source Names (DSNs)" on page 105

"Removing a DSN" on page 109

"Editing a DSN" on page 110

"Creating Database Connections (ASP Server)" on page 44

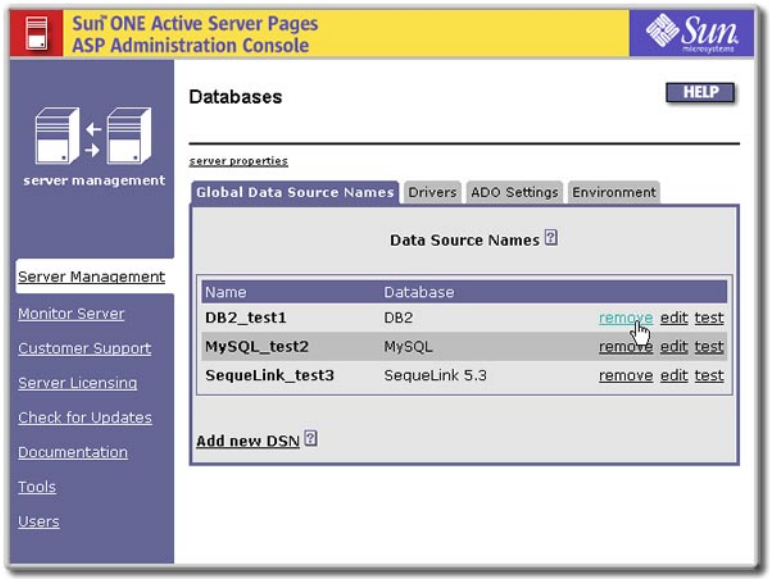
"Add/Edit DSNs" on page 100 (CLI)

Removing a DSN

A system DSN is removed from the ASP Server by using the Sun ONE Active Server Pages Administration Console. When you do this, the DSN can no longer be used in an ASP application to reference database connection information.

To remove a system DSN

1. Open the Administration Console (see "Accessing the Administration Console" on page 18).
2. On the **ASP Server** tab of the **Server Management** page, click **Databases**.
3. On the **Databases** page, in line with 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)" on page 105

"Adding a DSN" on page 106

"Editing a DSN" on page 110

"Testing a DSN" on page 111

"Creating Database Connections (ASP Server)" on page 44

"Delete DSNs" on page 100 (CLI)

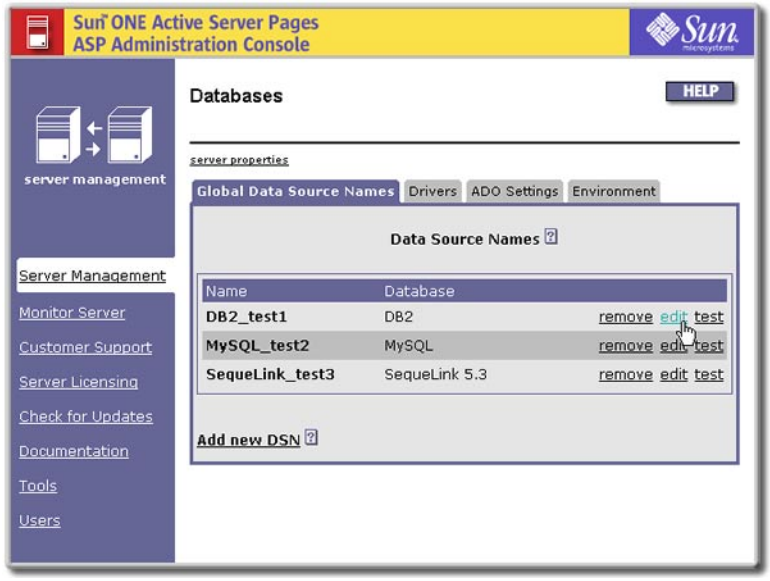
Editing a DSN

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.

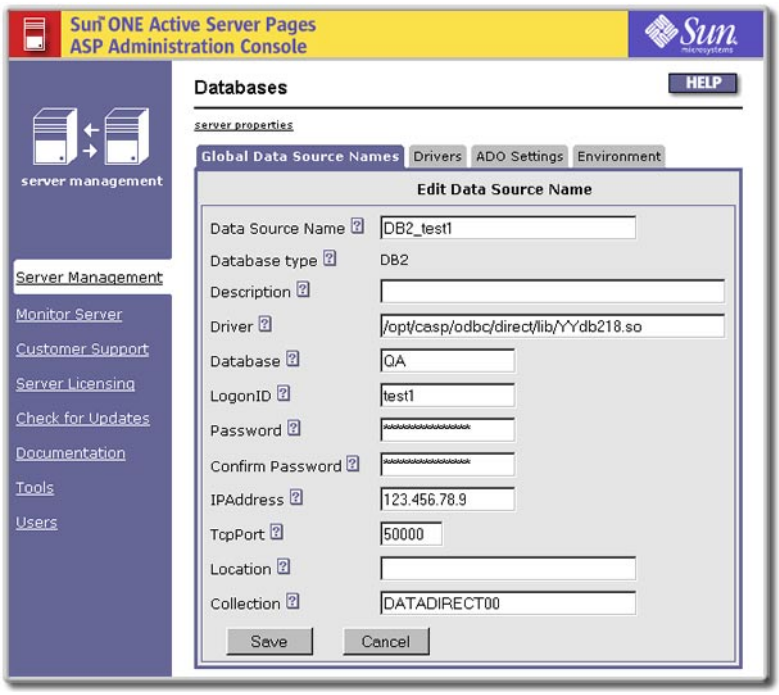
When you add a new DSN to the ASP Server, Sun ONE Active Server Pages automatically sets the correct parameters for the database. Changing these parameters can affect database performance, and is not recommended; the default settings are sufficient for most applications. Before editing database parameters, see the descriptions of required parameters for each ODBC driver in "Configuring Database Parameters" on page 115.

To edit system DSN information

1. Open the Administration Console (see "Accessing the Administration Console" on page 18).
2. On the **ASP Server** tab of the **Server Management** page, click **Databases**.
3. On the **Databases** page, in line with the DSN you want to edit, click **edit**.



The **Edit Data Source Name** page displays.



4. Edit the information as desired.
5. To save your changes, click **Save** and then click **Done**.

   - or -

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

See also:

"Configuring Data Source Names (DSNs)" on page 105

"Adding a DSN" on page 106

"Removing a DSN" on page 109

"Testing a DSN" on page 111

"Add/Edit DSNs" on page 100 (CLI)

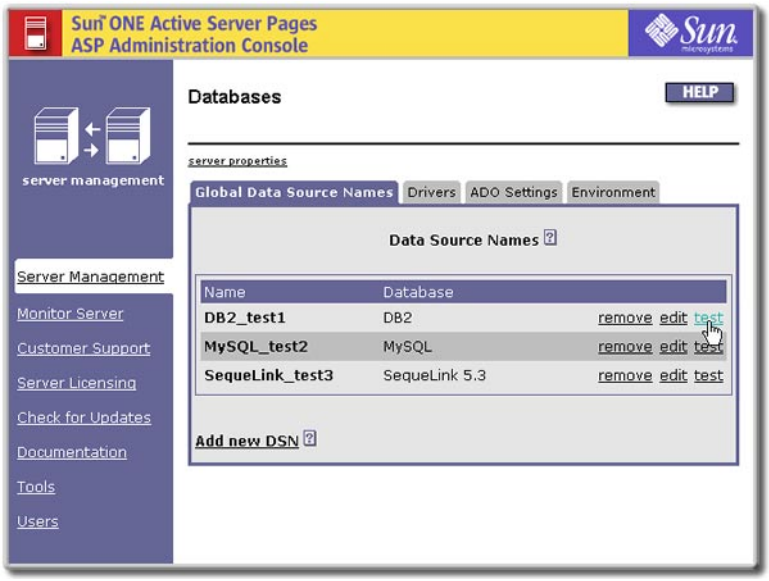
Testing a DSN

After adding a new system DSN or editing its parameters, use the following procedure to verify that the DSN is functioning correctly.

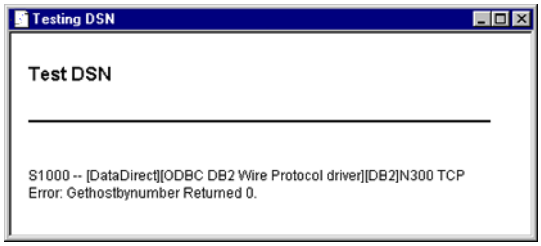
To test a system DSN

1. Open the Administration Console (see "Accessing the Administration Console" on page 18).

2. On the **ASP Server** tab of the **Server Management** page, click **Databases**.
3. On the **Databases** page, in line with the DSN you want to test, click **test**.



An advisory box opens, displaying information about the connection. If an error is listed, correct the problem and then retest the connection.



See also:

"Configuring Data Source Names (DSNs)" on page 105

"Adding a DSN" on page 106

"Editing a DSN" on page 110

"Removing a DSN" on page 109

Configuring the Database Environment

When you configure DSNs for Oracle and Informix databases (with client) for Sun ONE Active Server Pages, you also must specify additional environment information. For more information about the settings to use, contact your database administrator.

For more information about configuring DSNs, see "Configuring Data Source Names (DSNs)" on page 105.

In this section:

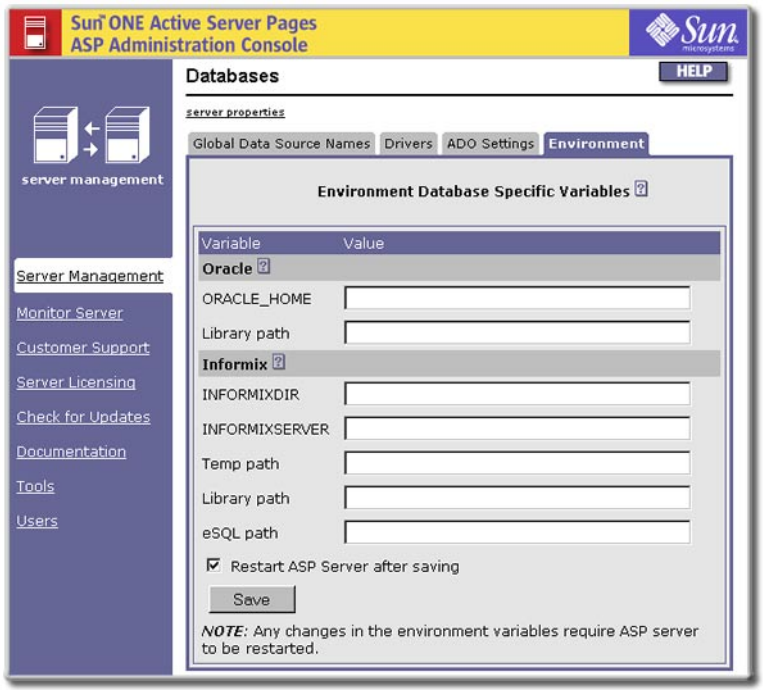
Setting Oracle Environment Variables

When you configure a DSN for an Oracle database (with client), you also must specify values for the **Oracle_Home** and **Library path** environment variables. For information about the values to set for these variables, contact your database administrator.

To set Oracle environment variables

1. Open the Administration Console (see "Accessing the Administration Console" on page 18).
2. On the **ASP Server** tab of the **Server Management** page, 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 ASP Server after saving** check box, and then click **Save**.

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

See also:

"Chapter 6, Configuring a Database" on page 103

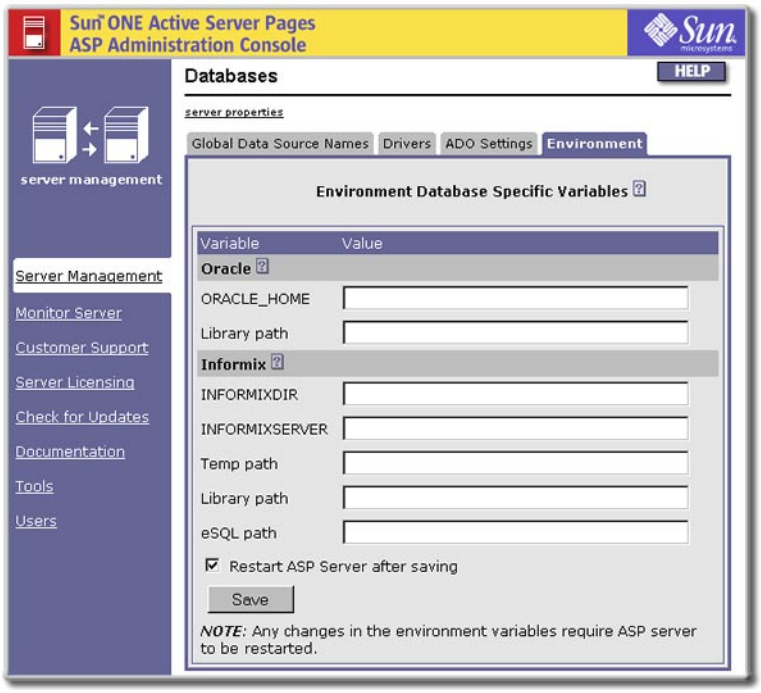
Setting Informix Environment Variables

When you configure a DSN for an Informix database (with client), you also must specify values for the **INFORMIXDIR**, **INFORMIXSERVER**, **Temp path**, **Library path**, and **eSQL path** environment variables. For information about the values to set for these variables, contact your database administrator.

To set Informix environment variables

1. Open the Administration Console (see "Accessing the Administration Console" on page 18).
2. On the **ASP Server** tab of the **Server Management** page, 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**, **INFORMIX-SERVER**, **Temp path**, **Library path**, and **eSQL path** boxes, type the desired values.
5. Select the **Restart ASP Server after saving** check box, and then click **Save**.

Note

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

See also:

"Chapter 6, Configuring a Database" on page 103

Configuring Database Parameters

To make it easier for Web developers to connect to a database from an ASP page, a system DSN for the database can be added to the ASP Server, as described in "Adding a DSN" on page 106. When you do this, Sun ONE Active Server Pages 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 configured by Sun ONE ASP. However, there might be times when you do need to edit them. This section provides reference information about the parameters that are configured for the ODBC drivers installed with Sun ONE ASP.

Note

Sun ONE Active Server Pages for UNIX and Linux installs the ODBC drivers for a number of databases (ODBC drivers are not installed with Sun ONE Active Server Pages for Windows). You can view the list of installed drivers from the Sun ONE ASP Administration Console, as described in "Viewing the List of ODBC Drivers" on page 104. Customer Support is provided only for ODBC drivers that are installed with Sun ONE ASP.

In this section:

"SequeLink Parameters" on page 128

"Sybase Parameters" on page 130

"Text Parameters" on page 131

See also:

"Supported in This Release" on page 5

"Chapter 6, Configuring a Database" on page 103

"Editing a DSN" on page 110

DB2 Parameters

The following table describes the DB2 (UDB, v7.1) database parameters available for configuring system DSNs, as they appear on the Sun ONE ASP Administration Console **New Data Source Name** and **Edit Data Source Name** pages. For more information, see "Configuring Data Source Names (DSNs)" on page 105.

Parameter	Explanation
DSN* **Data Source Name***	Name of the data source name (DSN) you are configuring. This must match the catalogued name of the DB2 database.
Description	Description of the DSN to distinguish it from others.
Database type*	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). This 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. This 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*	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. **Caution:** 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.

Parameter	Explanation
Password*	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. **Caution:** 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*	IP address for the database server (DB2).
TcpPort*	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	Package created by the DataDirect driver that reflects all parameters associated with a specific database (the parameters you specified). **Package** is displayed in the Sun ONE ASP 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:

"Chapter 6, Configuring a Database" on page 103

"Configuring Database Parameters" on page 115

dBASE Parameters

The following table describes the dBASE 5 database parameters available for configuring system DSNs, as they appear on the Sun ONE ASP Administration Console **New Data Source Name** and **Edit Data Source Name** pages. For more information, see "Configuring Data Source Names (DSNs)" on page 105.

Parameter	Explanation
DSN* **Data Source Name***	Name of the data source name (DSN) you are configuring.
Description	Description of the DSN to help distinguish it from others.

Parameter	Explanation
Database type*	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). This 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. This is a configurable field.
Database*	Path name of the directory in which the DBF files reside.
IntlSort	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:

"Chapter 6, Configuring a Database" on page 103

"Configuring Database Parameters" on page 115

Informix Parameters

This section describes the Informix database parameters available for configuring system DSNs as they appear in the Sun ONE ASP Administration Console. It specifies parameters for Informix 7 or 9 (with client) and Informix 2000 (without client)

In this section:

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 ONE ASP Administration Console **New Data Source Name** and **Edit Data Source Name** pages. For more information, see "Configuring Data Source Names (DSNs)" on page 105.

Note

This driver is not installed with Sun ONE Active Server Pages for Linux. Parameters for the Informix driver that is installed with Sun ONE ASP for

Linux are listed below in "Informix Parameters (Without Client): UNIX and Linux" on page 119.

Parameter	Explanation
DSN* **Data Source Name***	Name of the data source name (DSN) you are configuring.
Description	Description of the DSN to help distinguish it from others.
Database type*	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). This 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. This is a configurable field.
ServerName*	Name of the database server as it appears in the sqlhosts file.
HostName*	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	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. **Caution:** 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	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. **Caution:** 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:

"Informix Parameters (Without Client): UNIX and Linux" on page 119

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 ONE ASP Administration Console **New Data Source Name** and **Edit Data Source Name** pages. For more information, see "Configuring Data Source Names (DSNs)" on page 105.

Note

On Linux, only one Informix driver is listed in the Sun ONE Active Server Pages Administration Console.

Parameter	Explanation
DSN* **Data Source Name***	Name of the data source name (DSN) you are configuring.
Description	Description of the DSN to help distinguish it from others.
Database type*	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). This 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. This 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	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. **Caution:** 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	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. **Caution:** 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*	Name of the computer on which the Informix server resides.
PortNumber*	Port on which the database server is configured to listen. Ask your database administrator for this information.
ServerName*	Name of the database server as it appears in the sqlhosts file.

* Required parameters

See also:

"Informix Parameters (With Client): UNIX Only" on page 118

"Chapter 6, Configuring a Database" on page 103

"Configuring Database Parameters" on page 115

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 ONE ASP Administration Console **New Data Source Name** and **Edit Data Source Name** pages. For more information, see "Configuring Data Source Names (DSNs)" on page 105.

Parameter	Explanation
DSN* **Data Source Name***	Name of the data source name (DSN) you are configuring.
Description	Description of the DSN to help distinguish it from others.
Database type*	Indicates for which type of database you are configuring this DSN (Microsoft SQL Server).
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). This 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. This 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*	IP address of the SQL Server 7.0 or 2000 database server.
ServerPortNumber*	Port on which the SQL Server 7.0 or 2000 database server is configured to listen. The default is 1433.
LogonID	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. **Caution:** 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	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. **Caution:** 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:

"Chapter 6, Configuring a Database" on page 103

"Configuring Database Parameters" on page 115

MySQL Parameters

The following table describes the MySQL 3.23 database parameters available for configuring system DSNs, as they appear on the Sun ONE ASP Administration Console **New Data Source Name** and **Edit Data Source Name** pages. For more information, see "Configuring Data Source Names (DSNs)" on page 105.

Parameter	Explanation
DSN* **Data Source Name***	Name of the data source name (DSN) you are configuring.
Description	Description of the DSN to help distinguish it from others.
Database type*	Indicates for which type of database you are configuring this DSN (MySQL).
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). This 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. This is a configurable field.
Server*	IP address or name of the MySQL database server. If this field is empty, the server is assumed to be running on the local computer.
Port*	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	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. **Caution:** 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	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. **Caution:** 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	**UseCursorLib** is displayed in the Sun ONE ASP Administration Console only when you are editing an existing DSN, not adding a new one. 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:

"Chapter 6, Configuring a Database" on page 103

"Configuring Database Parameters" on page 115

Oracle Parameters

This section describes the Oracle database parameters available for configuring system DSNs as they appear in the Sun ONE ASP Administration Console. It specifies parameters for Oracle 7 and 8.05 (with client), and Oracle 8i (8.1.6 and 8.1.7) and 9i (without client).

In this section:

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 ONE ASP Administration Console **New Data Source Name** and **Edit Data Source Name** pages. For more information, see "Configuring Data Source Names (DSNs)" on page 105.

Note

This information applies to Solaris only.

Parameter	Explanation
DSN* **Data Source Name***	Name of the data source name (DSN) you are configuring.
Description*	Description of the DSN to help distinguish it from others.
Database type	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). This 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. This is a configurable field.
ServerName*	TNS name as defined in the tnsnames.ora file by the Oracle database client utility.

Parameter	Explanation
LogonID	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. **Caution:** 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	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. **Caution:** 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	**EnableDescribeParam** is displayed in the Sun ONE ASP Administration Console only when you are editing an existing DSN, not adding a new one. When this option is enabled (the check box is selected), all StoreProcedure arguments are returned as string types. This parameter is enabled by default.
ProcedureRetResults	**ProcedureRetResults** is displayed in the Sun ONE ASP Administration Console only when you are editing an existing DSN, not adding a new one. 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

See also:

"Oracle Parameters (Without Client)" on page 124

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 ONE ASP Administration Console **New Data Source Name** and **Edit Data Source Name** pages. For more information, see "Configuring Data Source Names (DSNs)" on page 105.

Parameter	Explanation
DSN* **Data Source Name***	Name of the data source name (DSN) you are configuring.
Description*	Description of the DSN to help distinguish it from others.
Database type	Indicates for which type of database you are configuring this DSN (Oracle).

Parameter	Explanation
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). This 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. This is a configurable field.
HostName	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	Port on which the database server is configured to listen. Ask your database administrator for this information.
SID	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	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. **Caution:** 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	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. **Caution:** 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	**EnableDescribeParam** is displayed in the Sun ONE ASP Administration Console only when you are editing an existing DSN, not adding a new one. 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	**ProcedureRetResults** is displayed in the Sun ONE ASP Administration Console only when you are editing an existing DSN, not adding a new one. When this option is enabled (the check box is selected), Oracle returns record sets from a StoredProcedure call. This parameter is enabled by default.

Parameter	Explanation
CatalogOptions	**CatalogOptions** is displayed in the Sun ONE ASP Administration Console only when you are editing an existing DSN, not adding a new one. 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**.
EnableStaticCursorsForLong Data	**EnableStaticCursorsForLongData** is displayed in the Sun ONE ASP Administration Console only when you are editing an existing DSN, not adding a new one. 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	**ApplicationUsingThreads** is displayed in the Sun ONE ASP Administration Console only when you are editing an existing DSN, not adding a new one. 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:

"Oracle Parameters (With Client)" on page 123

"Chapter 6, Configuring a Database" on page 103

"Configuring Database Parameters" on page 115

PostgreSQL Parameters

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 ONE ASP Administration Console **New Data Source Name** and **Edit Data Source Name** pages. For more information, see "Configuring Data Source Names (DSNs)" on page 105.

Parameter	Explanation
DSN* **Data Source Name***	Name of the data source name (DSN) you are configuring.
Description	Description of the DSN to help distinguish it from others.

Parameter	Explanation
Database type*	Indicates for which type of database you are configuring this DSN (PostgreSQL).
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). This 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. This is a configurable field.
ServerName*	IP address of the PostgreSQL database server. If this field is empty, the server is assumed to be running on the local computer.
Port*	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	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. **Caution:** 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	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. **Caution:** 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	**UseCursorLib** is displayed in the Sun ONE ASP Administration Console only when you are editing an existing DSN, not adding a new one. 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:

"Chapter 6, Configuring a Database" on page 103

"Configuring Database Parameters" on page 115

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 ONE ASP Administration Console **New Data Source Name** and **Edit Data Source Name** pages.

Parameter	Explanation
DSN* **Data Source Name***	Name of the data source name (DSN) you are configuring. It must match the entry for **ServerDataSource** (below), which is the DSN created on the SequeLink server. For more information, see "Configuring SequeLink" on page 128.
Description*	Description of the DSN to help distinguish it from others.
Database type	On the **New Data Source Name** page, select **SequeLink 5.3** 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 5.3** appears in this field.
Driver*	On the **New Data Source Name** page, this is the name of the database driver configured for this DSN (SequeLink). This 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. This is a configurable field.
Host	IP address of the SequeLink server.
Port	Port the SequeLink server is listening on. Default is 19996.
ServerDataSource*	Name of the DSN configured on the SequeLink server. For more information, see "Configuring SequeLink" on page 128.

* Required parameters

See also:

"Chapter 6, Configuring a Database" on page 103

"Configuring Database Parameters" on page 115

Configuring SequeLink

Sun ONE Active Server Pages includes the client portion of DataDirect SequeLink 5.3, 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. Create a data source on the Windows machine.
2. Install and configure the SequeLink server software on the database server. The software can be downloaded from the Sun ONE ASP Web site at:

   http://developer.chilisoft.com/downloads/register2.asp?target=SEQUELINK

   Run setup.exe /v"IPE=No" and install all defaults.

   Note: Be sure to install SLSocket. For information about configuring the SequeLink server, see the following procedure.
3. Add a SequeLink DSN by using the Sun ONE ASP Administration Console, as described in "Adding a DSN" on page 106. When you do this, be sure to use the DSN name that you create in the following procedure.

To configure SequeLink

1. On the database server to which SequeLink Server 5.3 is installed, create a DSN using Windows' administrative tools (**Start** > **Settings** > **Control Panel** > **Administrative Tools** > **Data Sources (ODBC)**.
2. Start SequeLink Server 5.3 for ODBC Sockets (**Start** > **SequeLink Server 5.3 for ODBC Sockets** > **SequeLink Manager Snapin**.

   Note: To perform this procedure, SLSocket must be installed.
3. Connect to the SequeLink Service if you're not already connected, and then expand **SequeLink 5.3 Manager**.
4. Expand **SequeLink Services** > **SLSocket53** > **Configuration**.
5. Right-click **Data Source Settings**, and then select **New** > **Data source**.
6. Name the new data source by typing the name of the DSN you created in step 1 (for example, "test"), expand the node, and then click the **Advanced** folder.
7. Double-click **DataSourceSOCODBCConnStr**.
8. In the **Value** box, change the value to the connection string you created in step 1 (for example, "DSN=test"), and then click **OK**.
9. Expand the **User Security** node, and then double-click **Data Source Logon Method**.
10. Select **OSIntegrated**.
11. Configure the SequeLink DSN, as described in "Adding a DSN" on page 106. You will be asked to specify the **ServerDataSource**. This is the name of the DSN you just configured (for example, "test").

See also:

"Chapter 6, Configuring a Database" on page 103

"SequeLink Parameters" on page 128

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 ONE ASP Administration Console **New Data Source Name** and **Edit Data Source Name** pages. For more information, see "Configuring Data Source Names (DSNs)" on page 105.

Parameter	Explanation
DSN* **Data Source Name***	Name of the data source name (DSN) you are configuring.
Description	Description of the DSN to help distinguish it from others.
Database type*	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). This 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. This is a configurable field.
Server*	IP address or name of the Sybase database server. If this field is empty, the server is assumed to be running on the local computer.
Port*	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*	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. **Caution:** 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*	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. **Caution:** 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:

"Chapter 6, Configuring a Database" on page 103

"Configuring Database Parameters" on page 115

Text Parameters

The following table describes the Text parameters available for configuring system DSNs, as they appear on the Sun ONE ASP Administration Console **New Data Source Name** and **Edit Data Source Name** pages. For more information, see "Configuring Data Source Names (DSNs)" on page 105.

Parameter	Explanation
DSN* **Data Source Name***	Name of the data source name (DSN) you are configuring.
Description	Description of the DSN to help distinguish it from others.
Database type*	Indicates for which type of database you are configuring this DSN.
Driver*	This is the installed ODBC driver specified for this DSN.
Database*	Directory in which the text files are stored. If left empty, the current working directory is used.
TableType	Default table type (Comma, Tab, Character, Fixed, or Stream). The Text driver supports five table-types: comma-separated, tab-separated, character-separated, fixed length, and stream. The default table type is used when creating a new table, and opening an undefined table.

***** Required parameters

See also:

"Chapter 6, Configuring a Database" on page 103

"Configuring Database Parameters" on page 115

Configuring ADO Connections

ADO (ActiveX Data Objects) is the Microsoft standard for database access. Sun ONE Active Server Pages provides an ADO control, which you can configure by using the Sun ONE ASP Administration Console. For more information about ADO, see "Creating Database Connections (ASP Server)" on page 44 and "ADO Component Reference" on page 301.

In this section:

Setting the ADO Connection Pool Size

Sun ONE Active Server Pages supports database connection pooling, which improves the performance of applications that rely heavily on database operations. With

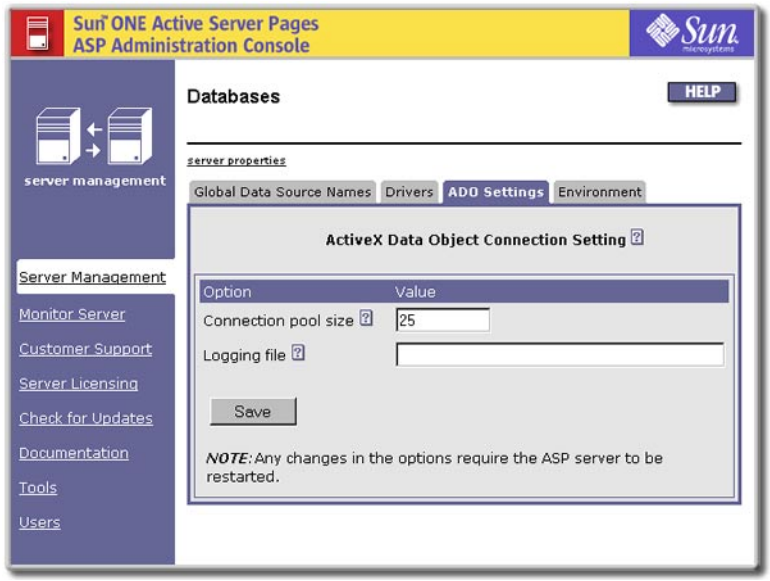
connection pooling, rather than opening and closing a database connection for each individual request, Sun ONE ASP uses a connection that is already open.

Sun ONE ASP uses an ADO control to provide database connectivity. The ADO connection pool size parameter is set in the Sun ONE ASP Administration Console. The default ADO connection pool size is 25, which can be increased or decreased according to your requirements. There is no maximum number of connections that can be pooled. Setting this to 0 (zero) disables connection pooling.

To set the ADO connection pool size

1. Open the Administration Console (see "Accessing the Administration Console" on page 18).
2. On the **ASP Server** tab of the **Server Management** page, 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" on page 72

Enabling and Disabling ADO Logging

Sun ONE Active Server Pages uses an ADO (ActiveX Data Objects) control to provide database connectivity. Logging for ADO is enabled from the Sun ONE ASP Administration Console by providing an absolute path name for the log file. When you do this, Sun ONE ASP creates the log file in the directory specified, and begins logging to it. To disable logging, simply delete the path name of the log file.

ADO logging will not be functional if **Inherit user security** is set to **yes**. For information about this setting, see "Setting the Security Mode" on page 57.

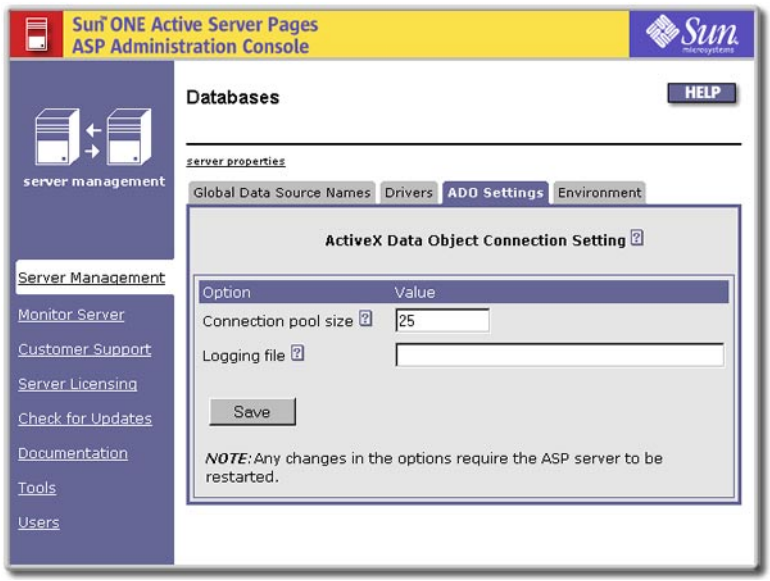
Caution

ADO logging should be used for diagnostic purposes only, and should not be enabled when running Sun ONE ASP on a production server.

To enable or disable ADO logging

1. Open the Administration Console (see "Accessing the Administration Console" on page 18).
2. On the **ASP Server** tab of the **Server Management** page, 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.

   To disable ADO logging, leave the **Logging file** box empty, or delete existing text.

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:

"Configuring ADO Connections" on page 131

7 Using Database Tools

Sun™ ONE Active Server Pages for Solaris and Linux includes two database tools: Sun ONE ASP Database Publisher (Database Publisher), and Sun ONE ASP Database Management System for MySQL (DBMS).

Database Publisher is a client/server application that enables a Microsoft Access database running on Windows to be published to a MySQL database running on UNIX or Linux (with Sun ONE ASP installed). DBMS is a database administration tool that enables MySQL databases to be administered from a browser-based administration console instead of strictly from the command line.

This chapter describes how to administer and use these tools.

In this chapter:

Database Publisher

Migrating Web sites from Windows to UNIX or Linux is problematic if existing Web applications use Microsoft Access databases. The contents of Access database files must be migrated to a database supported on UNIX or Linux, but no ODBC drivers exist that can read or write Access files on those platforms. Sun ONE Active Server Pages Database Publisher (Database Publisher) addresses this problem.

Database Publisher is a client/server application that enables a Microsoft Access database running on Windows to be published to a MySQL database running on UNIX or Linux (with Sun ONE ASP installed). The application captures the schema and data of an Access database on a Windows machine, and then transmits those contents in the bodies of HTTP requests to a MySQL database on a UNIX or Linux machine. The contents of the Access database are read from the HTTP requests and used to create a copy of the original Access database on the MySQL database server.

Database Publisher is administered from the Sun ONE ASP Administration Console, which is installed with Sun ONE Active Server Pages. The client component is an executable file that is downloaded from the Web and installed on the Windows machine. Database Publisher is a wizard-like application that guides you through each step of the publishing process.

This section describes how to administer and use Database Publisher, and assumes a working knowledge of both Access and MySQL.

In this section:

"Using the Database Publisher Wizard" on page 138

Administering Database Publisher

Database Publisher is administered from the Sun ONE ASP Administration Console. Administrators use the Administration Console to enable Database Publisher, specify an authorization key that unlocks the application for use, and specify global Create privileges.

To make Database Publisher available for users, administrators must do the following:

- Enable the Database Publisher application for use in the Sun ONE ASP Administration Console.
- Create a database account for each user of Database Publisher, and furnish each user with a user ID and password for the MySQL server.
- Choose an authorization key and provide it to users. This key unlocks the application for use (the key will be the same for all users).
- Specify global Create privileges for the MySQL server. If users are not allowed to create a new database, a database must be supplied on the MySQL server for which users have all permissions.

Note

All database security is governed by the privileges assigned to the user account by the MySQL database administrator. For more information about MySQL security, see documentation on the MySQL Web site pertaining to the privilege system (go to http://www.mysql.com/documentation).

To administer Database Publisher

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

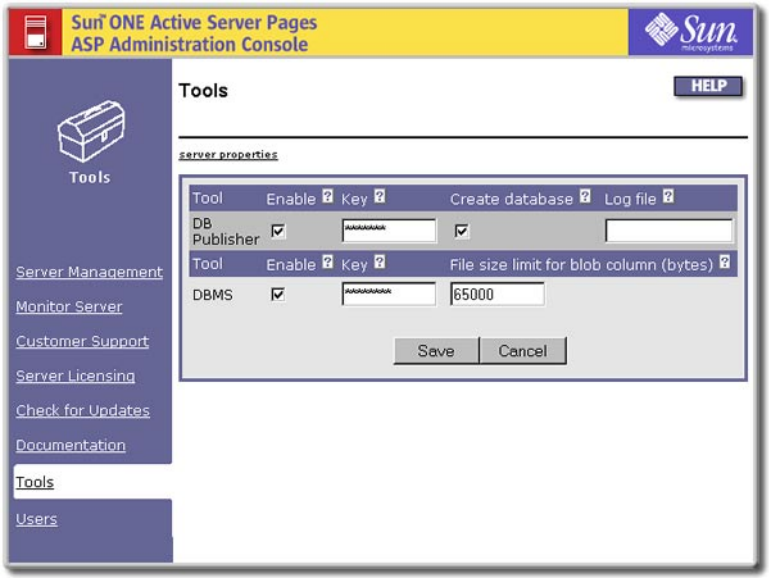
   http://[HOSTNAME]:[PORT]

   where [HOSTNAME] is the hostname of the Web server configured to run with Sun ONE ASP, 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 **Tools**.

   The **Tools** page displays.



3. To enable or disable Database Publisher, select or clear the **Enable** box for **DB Publisher** (Database Publisher is enabled by default).
4. In the **Key** box for Database Publisher, specify the authorization key that unlocks the application for use. This authorization key must be supplied to each user of the application. The key is configured as "password" by default. A new key should be chosen as soon as possible.
5. Select or clear the **Create database** box to specify global Create privileges:
   - If this box is selected, client-side users with appropriate privileges can create a new database on the MySQL server to which to publish an Access database.
   - If this box is cleared, all users are barred from creating a new database on the MySQL server, regardless of other user privileges. If users are not allowed to create a new database, a database must be supplied on the MySQL server for which users have all permissions.
6. In the **Log file** box, specify the path for application logging information. If this box is empty, logging for the application will be disabled.
7. Click **Save** to save your changes, or **Cancel** to revert to the settings that were last saved.

See also:

"Installing Database Publisher" on page 138

"Using the Database Publisher Wizard" on page 138

Installing Database Publisher

The client component of Database Publisher is an executable program installed on the Windows machine. The executable (.exe) file is downloaded from the Sun ONE ASP download site.

To install Database Publisher

1. Download the PublisherWizard_setup.exe file from the following location:

   http://developer.chilisoft.com/downloads

2. Open the file.
3. Perform the installation as prompted by the setup program.

Note

To uninstall Database Publisher, use the Add/Remove functionality in Windows.

See also:

"Using the Database Publisher Wizard" on page 138

Using the Database Publisher Wizard

The Database Publisher wizard is used to publish an Access database to MySQL. The wizard is installed on the Windows (client) machine as described in "Installing Database Publisher" on page 138, and walks you through each step of the publishing process. In general, you will take the following steps:

- Select the Access file you want to publish.
- Verify that table, index, and column names in the Access database are valid in MySQL, and fix any names that are invalid.
- Provide an authorization key and other information needed to unlock the application for use.
- Specify the destination (MySQL) database to which you want to publish the Access database.
- Review conflicting (duplicate) tables in the Access and MySQL databases, if any, and take desired actions.
- Publish the Access database to MySQL.

Before you begin, make sure you have the following information:

- Name of the Microsoft Access file you want to publish.
- Name and port number for the Web server to which the Sun ONE ASP Administration Console is installed (used to verify the authorization key). If a proxy server is in use, you must know the name and port number for the proxy server.
- Authorization key (unlocking password) for Database Publisher.

- Server name and port number for the destination (MySQL) database.
- User ID and password for the MySQL server.

This section describes how to perform each step of the publishing process.

In this section:

See also:

"Installing Database Publisher" on page 138

Opening the Database Publisher Wizard

Use the following procedure to open the Database Publisher wizard. The wizard is used to publish an Access database to MySQL. To use the wizard, Database Publisher must be enabled in the Sun ONE ASP Administration Console (see "Administering Database Publisher" on page 136).

To open the Database Publisher wizard

- Double-click the Database Publisher icon on your desktop, or choose **Start** > **Programs** > **Sun ONE ASP** > **Database Publisher > SOA DB Publisher**.

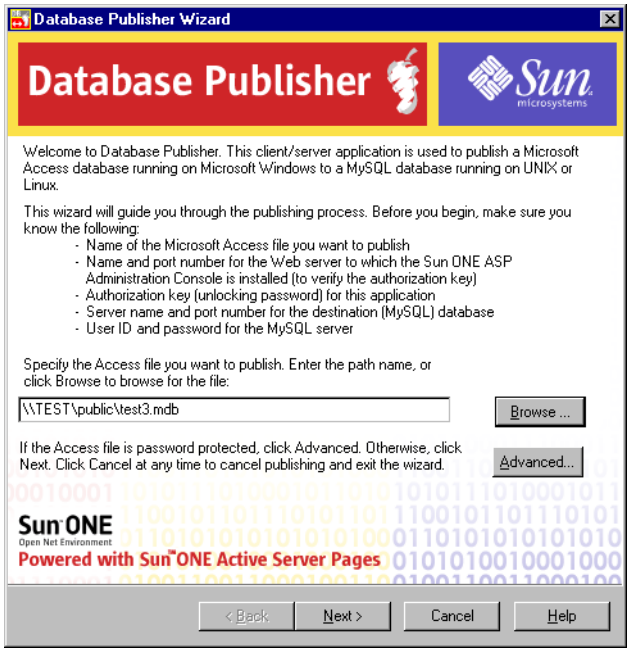
  The Database Publisher wizard opens, and you are ready to proceed with publishing.

See also:

"Using the Database Publisher Wizard" on page 138

Selecting the Access File

The first step in the publishing process is to specify the Access file you want to publish. This specification is made on the first screen to display when you open the Database Publisher wizard.



To select the Access file

1. Open the Database Publisher wizard (see "Opening the Database Publisher Wizard" on page 139).
2. Specify the Access file to be published.

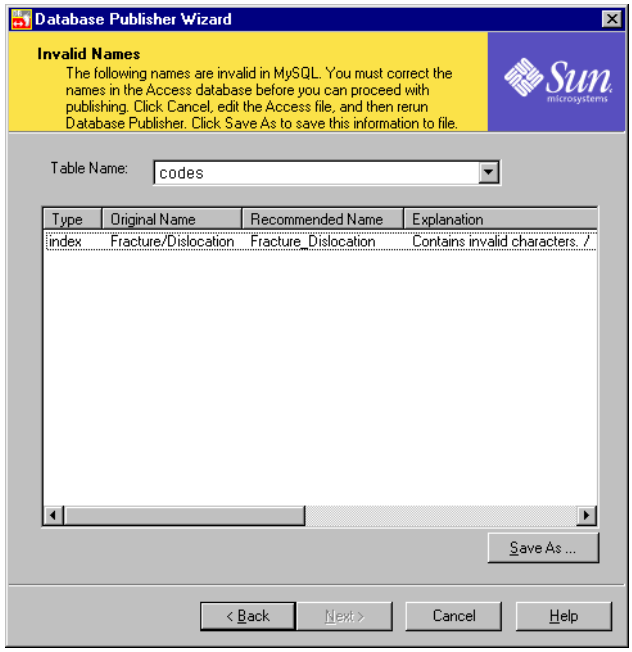
   If the Access file is password protected, click **Advanced**. In the **File Password** dialog that displays, select the check box, type the database password in the **Password** text box, and then click **OK**.
3. Click **Next** to proceed with publishing, or **Cancel** to exit the wizard.

See also:

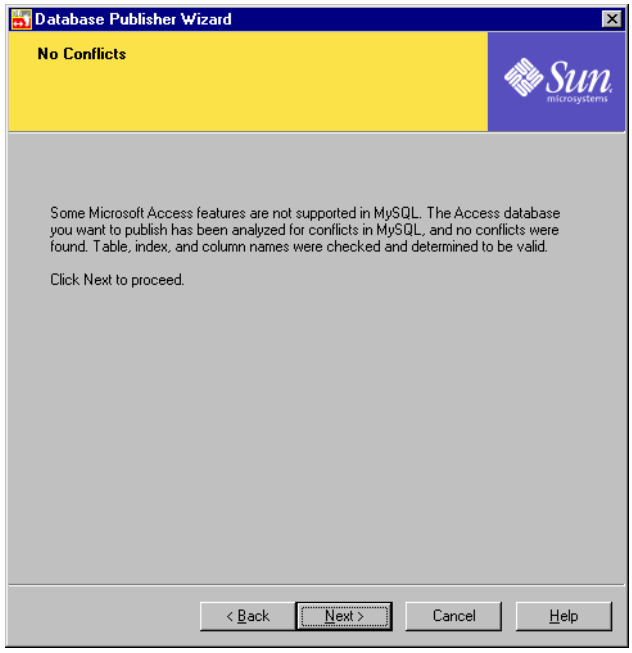
"Using the Database Publisher Wizard" on page 138

Resolving Invalid Names

Table, column, and index names that are valid in Access might be invalid in MySQL. Database Publisher detects invalid names and lists them on the **Invalid Names** screen in the wizard, providing an explanation of why the names are invalid and suggestions for correction. The invalid names must be corrected in the Access database before the database can be published to MySQL.



If no invalid names are detected, the **No Conflicts** screen displays in the wizard, and you can click **Next** to proceed with publishing.



The following procedure lists the general steps for resolving invalid names.

To resolve invalid names

1. On the **Invalid Names** screen, click **Save As** to save the information to a file.

2. Click **Cancel** to exit the Database Publisher wizard.
3. Correct the Access database.
4. Rerun the Database Publisher wizard (see "Opening the Database Publisher Wizard" on page 139).

See also:

"Using the Database Publisher Wizard" on page 138

Verifying the Authorization Key

Before you can publish a database, you must provide a valid authorization key. This key unlocks the application for use, and is configured and provided by your Sun ONE ASP administrator (see "Administering Database Publisher" on page 136). You cannot publish a database without a valid authorization key. If you do not have this key, contact your administrator.

The authorization key is specified on the **Verify Authorization Key** screen in the Database Publisher wizard. You must also provide information about the Web server to which the Sun ONE ASP Administration Console is installed (this information is used for key verification).

To verify the authorization key

1. On the **Verify Authorization Key** screen, specify information about the Web server to which the Sun ONE ASP Administration Console is installed:
   - In the **ASP Admin Server** box, specify the hostname for the Web server.
   - In the **Port** box, specify the port number for the Web server.

- If a proxy server is in use, click **Use Proxy**. In the **Proxy Server** dialog that displays, select the **Use HTTP Proxy** check box, provide the host-name and port number for the proxy server, and then click **OK** to return to the **Verify Authorization Key** screen (or **Cancel** to exit this dialog).

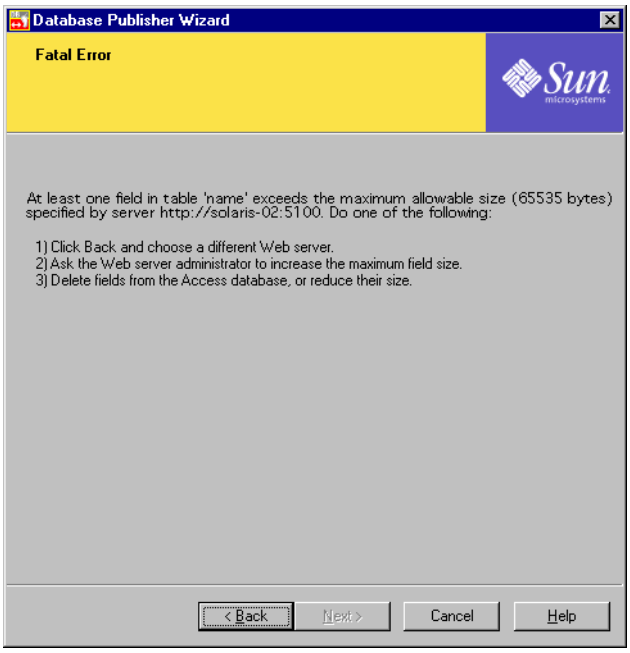
2. In the **Key** box, specify the authorization key. If you do not have this key, contact your administrator.
3. Click **Next** to proceed with publishing, **Cancel** to exit the wizard, or **Back** to return to a previous screen.

See also:

"Using the Database Publisher Wizard" on page 138

Fatal Error Screen

The **Fatal Error** screen pertains to the file size limit for blob columns. It displays following the **Verify Authorization Key** screen if the data passed to a database exceeds the maximum long field length set in the Sun ONE ASP configuration file, casp.cnfg. The `maxlongfieldlength` setting in casp.cnfg specifies the maximum long field length in bytes, and is set to 65535 by default.



To proceed with publishing, do one of the following:

- Click **Back** in the wizard and choose a Web server for which this value is appropriately set.
- Delete fields from the Access database, or reduce their size.
- Ask your Sun ONE ASP administrator to increase the `maxlongfieldlength` value in casp.cnfg (see the `[ADO]` keyword).

Caution

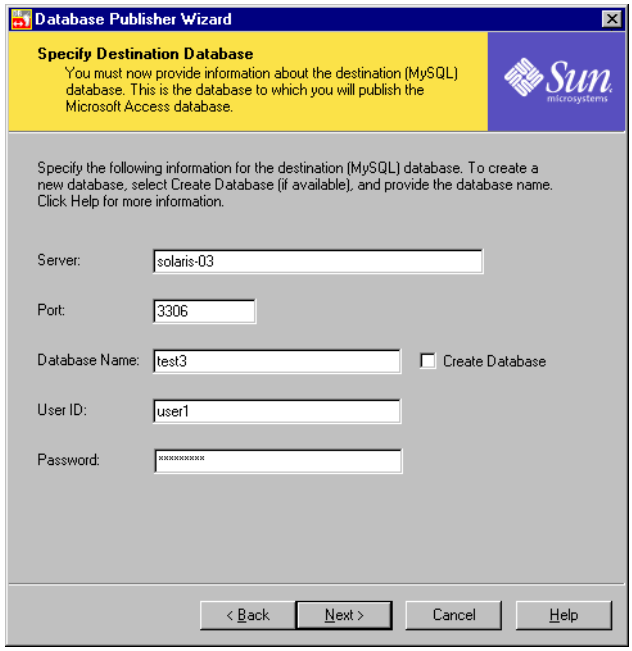
Setting `maxlongfieldlength` to an extreme value (greater than 50 MB, for example) could cause the MySQL database server to crash and the Database Publisher application to fail.

See also:

"Using the Database Publisher Wizard" on page 138

Specifying the Destination Database

You must provide information about the MySQL database to which the Access database will be published. This information is provided on the **Specify Destination Database** screen in the Database Publisher wizard.



To specify the destination database

1. On the **Specify Destination Database** screen, in the **Server** box, specify the hostname or IP address for the MySQL server.
2. In the **Port** box, specify the port for the MySQL server (3306 by default).
3. In the **Database Name** box, specify the name of the destination (MySQL) database.

   If you receive an error message pertaining to an "unknown database," that means the database you have specified does not exist on the MySQL server and must be created. See the following note.

Note

If you have appropriate privileges for the database server, and **Create database** has been enabled by the administrator on the **Tools** page in the Sun ONE ASP Administration Console, the **Create Database** check box is also available on this screen. Select the check box to create a new database to which to publish, and specify the name of the new database in the **Database Name** box.

4. In the **User ID** box, specify the user name for the MySQL database server, as provided by the database administrator.
5. In the **Password** box, specify the password for the MySQL database server, as provided by the database administrator.
6. Click **Next** to proceed with publishing, **Cancel** to exit the wizard, or **Back** to return to a previous screen.

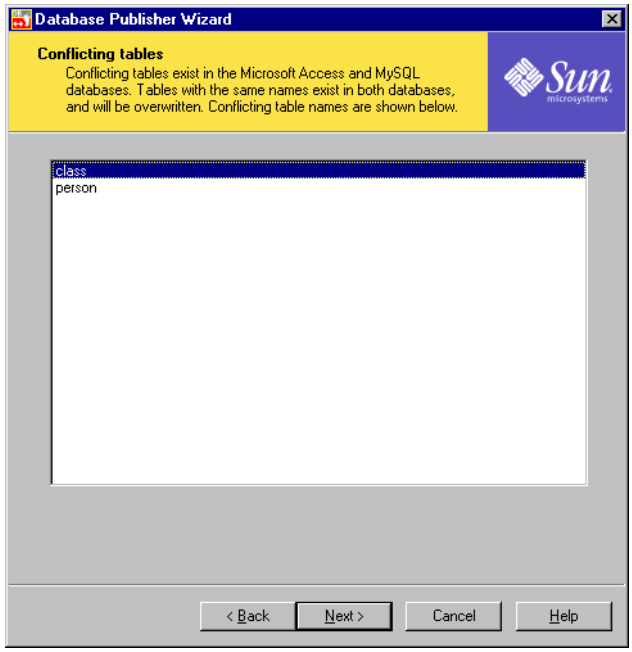
See also:

"Using the Database Publisher Wizard" on page 138

Conflicting Tables

Database Publisher creates tables in the MySQL database with the same names as those in the Access database, and then populates those tables with the source (Access) data. If duplicate table names exist, the table with the same name in the MySQL database will be replaced with the table in the Access database.

The **Conflicting tables** screen in the Database Publisher wizard lists any such conflicts.



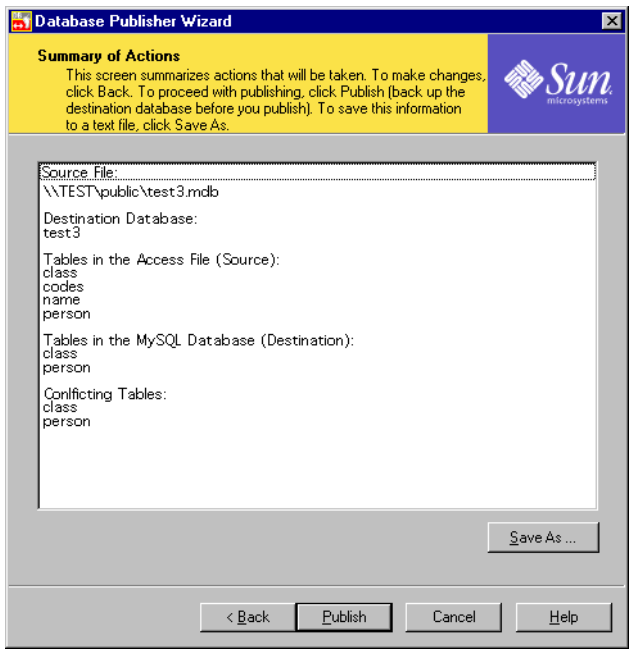
This screen simply advises you that the MySQL tables will be replaced. If you click **Next** on this screen, you will be asked if you want to continue, even though the conflicting tables in MySQL will be replaced during publishing.

See also:

"Using the Database Publisher Wizard" on page 138

Publishing the Database

The **Summary of Actions** screen in the Database Publisher wizard is the final screen to display before publishing the Access database to MySQL.



This screen summarizes all actions that will be taken during publishing, and is your last chance to cancel the process and cleanly exit the publishing wizard before data in the MySQL database has been altered.

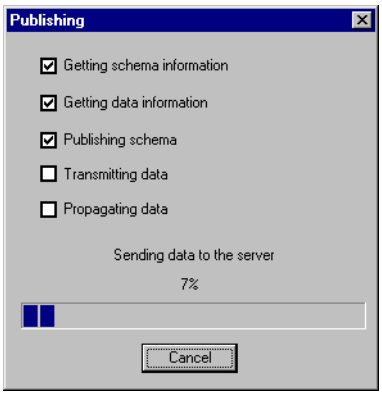
After a database is published, an association is created between the original Windows file name and a connection string that connects to the new database. The association is used by the ADO control.

Caution

Before publishing, be sure to back up the MySQL database to which you are publishing.

To publish the database

1. Verify all actions listed on the **Summary of Actions** screen. To make changes, click **Back** to return to a previous screen. If desired, click **Save As** to save this information to a file.
2. Start or cancel publishing:
   - To proceed with publishing, click **Publish**. Publishing begins and the **Publishing** dialog displays, showing progress.



   - or -
   - Click **Cancel** to exit the wizard and cancel publishing before data in the MySQL destination database has been altered. Do not wait to click **Cancel** in the **Publishing** dialog after publishing has begun. That action will have unpredictable results for the destination (MySQL) database.

See also:

"Using the Database Publisher Wizard" on page 138

DBMS

The Sun ONE Active Server Pages Database Management System for MySQL (DBMS) is a database administration system for MySQL databases. The tool enables MySQL databases to be administered from a browser-based administration console instead of strictly from the command line. This user-friendly approach greatly simplifies the administration of MySQL databases, allowing a graphical user interface (GUI) to be used to perform database administration.

The DBMS application is installed with Sun ONE Active Server Pages and is administered from the Sun ONE ASP Administration Console, where it runs as an ASP application. A client Web browser is used to access DBMS, and to connect to and administer a database on the MySQL database server.

This section describes how to administer and use the DBMS application, and assumes a working knowledge of MySQL. It does not provide detailed information about MySQL, or about relational database concepts in general. For specific information about the installation, configuration, and use of MySQL, please refer to MySQL documentation at the following URL:

http://www.mysql.com/documentation

Please note the following:

- You cannot use this tool to create a new database. It is assumed that a MySQL server is available to which you can connect, and that a database exists on that server.
- The information displayed and the actions you can take in DBMS are governed solely by the privileges granted to you by the Sun ONE ASP and MySQL administrators.
- If information does not display as you think it should when using DBMS (for example, you update a table and your changes are not reflected in the user interface), you may need to refresh your browser by right-clicking in the pane and then clicking **Refresh**.

In this section:

Administering DBMS

The DBMS application is administered from the Sun ONE ASP Administration Console. Administrators use the Administration Console to enable DBMS for use, specify an authorization key that unlocks the application, and specify the blob file-size limit.

To make DBMS available for users, Sun ONE ASP administrators must do the following:

- Enable the DBMS application in the Sun ONE ASP Administration Console.
- Choose an authorization key and provide it to users. This key unlocks the application for use (the key will be the same for all users).

  The session timeout is set to 20 minutes by default. If the session expires, users will be prompted to log back in. This timeout setting is configured in the global.asa file for DBMS. Likewise, if the Sun ONE ASP administrator changes the authorization key, users will be prompted to log back in with the new key.
- Specify the blob file-size limit.
- Provide users with the URL needed to access DBMS. Typically this will be:

  //[HOSTNAME]/dbms

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

The DBMS application cannot be used to create a new database. It is assumed that a MySQL server is available to which users can connect, and that a database exists on that server.

Note

Database security is governed solely by privileges assigned to the user account by the MySQL database administrator. For more information about MySQL security, see documentation on the MySQL Web site pertaining to the privilege system.

To administer DBMS

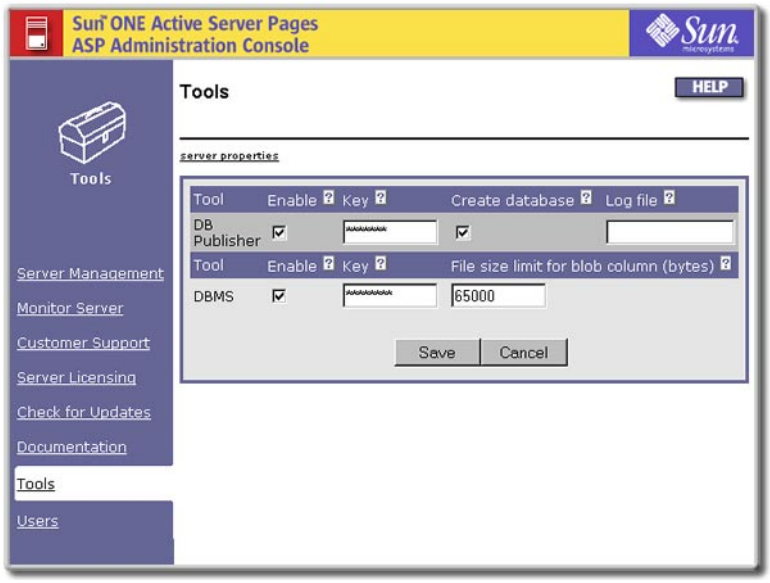
1. Open the Sun ONE ASP Administration Console by using the following URL:

   http://[HOSTNAME]:[PORT]

   where [HOSTNAME] is the hostname of the Web server configured to run with Sun ONE ASP, 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 **Tools**.

   The **Tools** page displays.



3. To enable or disable DBMS, select or clear the **Enable** box for DBMS (DBMS is enabled by default).
4. In the **Key** box for DBMS, specify the authorization key that unlocks the application for use. This authorization key must be supplied to each user of the application. The key is configured as "password" by default. A new key should be chosen as soon as possible.
5. In the **File size limit for blob column (bytes)** box, specify the blob file-size limit. This setting limits the size of files users can upload into their databases.

   This value defaults to the smaller of the following two values:

   - The `maxlongfieldlength` parameter for ADO in the Sun ONE ASP configuration file, casp.cnfg, which specifies the maximum long field length in bytes (see the `[ADO]` keyword). This value is set to 65535 by default. If the data passed to a database exceeds this limit, ADO will throw an error.
   - The **Max. Transfer Size (Bytes)** setting on the **Components** page in the Sun ONE ASP Administration Console. This setting specifies the maximum transfer size (in bytes) that can be uploaded using the Chili!Upload component. This value is set to 1000000 by default.

   Caution

   Severe problems will result if either of these values is set extremely high (such as 50 MB for `maxlongfieldlength` or 30 MB for **Max. Transfer Size**).

6. Click **Save** to save your changes, or **Cancel** to revert to the settings that were last saved.

See also:

"Accessing DBMS" on page 151

Accessing DBMS

DBMS allows you to administer MySQL databases from a GUI accessed with a client browser. To use DBMS, the tool must be enabled in the Sun ONE ASP Administration Console. Users logging in to the DBMS application will be asked to provide an authorization key.

The authorization key unlocks DBMS for use, and is configured by your Sun ONE ASP administrator in the Sun ONE ASP Administration Console (see "Administering DBMS" on page 149). You cannot use the application without a valid authorization key. If you do not have this key, contact your administrator.

The session timeout is set to 20 minutes by default. If your DBMS session expires, you will be prompted to log back in. Likewise, if the Sun ONE ASP administrator changes the authorization key, you will be prompted to log back in with the new key. Contact your Sun ONE ASP administrator if you feel the session timeout is too short, or if you are having trouble logging in.

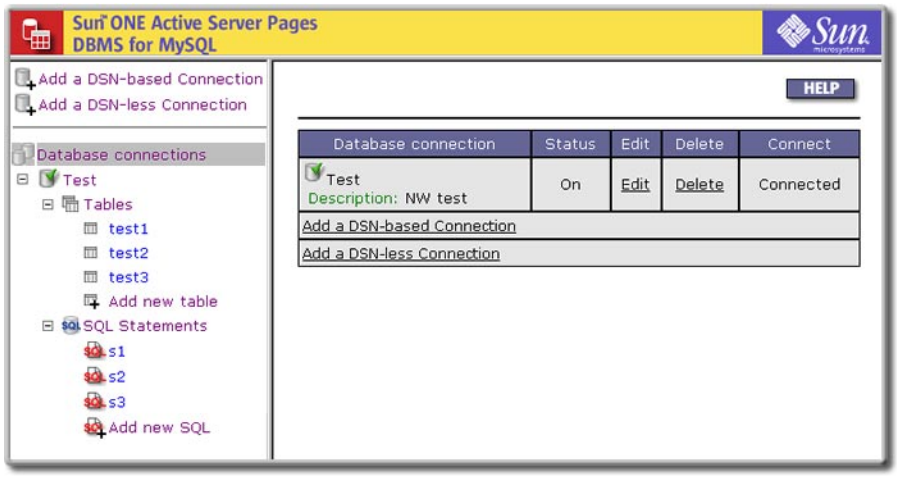
Caution

Do not use this application from a public computer. Doing so could expose your information to other users of the shared computer. If you must use a public computer, it is strongly recommended that you delete database connections before closing the application or browser, as described in "Deleting a DSN-based Connection (DBMS)" on page 158 and "Deleting a DSN-less Connection (DBMS)" on page 164.

To access DBMS

- Access DBMS using the information provided by your Sun ONE ASP administrator.

  The DBMS user interface defaults to the "last visited" view, and displays two primary panes. The left pane is the navigation pane and provides a tree view of the database structure hierarchy. The right pane is the content pane and displays node properties, user forms, or query execution results. Connection nodes in the left pane can be expanded by clicking text, or by clicking the plus (+) sign.



If you are using DBMS for the first time, your first step is to create and establish a connection to the MySQL database, as described in "Connecting to a Database (DBMS)" on page 152.

Note

If information does not display as you think it should when using DBMS (for example, you update a table and your changes are not reflected in the user interface), you may need to refresh your browser by right-clicking in the pane and then clicking **Refresh**.

See also:

"Connecting to a Database (DBMS)" on page 152

DBMS Conventions

The Sun ONE ASP DBMS application is used to administer MySQL databases only. Syntax, column types, and naming conventions adhere to MySQL specifications. When using DBMS, you must use MySQL conventions or errors will result.

This documentation is not intended to serve as a MySQL language reference. For detailed information about MySQL conventions and rules, consult the documentation on the MySQL Web site at the following URL (particularly the language reference section):

http://www.mysql.com/documentation

Connecting to a Database (DBMS)

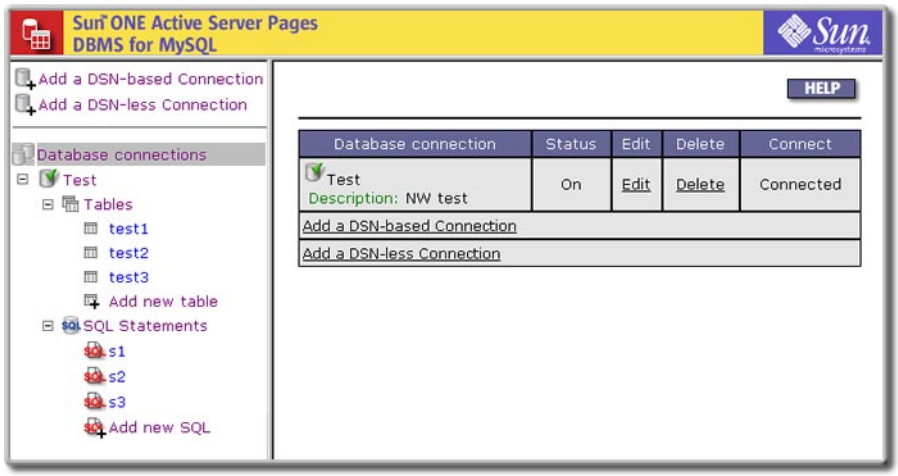
MySQL databases are administered via database connections configured in DBMS. These connections are comprised of a title and the actual connection string used to connect to a database, and can be configured using DSN-based or DSN-less

connection strings (data source names are collections of information required for connecting to a specific database).

Your first step in using DBMS is to create and establish a connection to a MySQL database. Once a connection is successfully established, DBMS can then be used to perform such tasks as creating and designing tables, adding and editing data, and querying the database using SQL statements. The tasks you can perform are determined by privileges granted by the database administrator.

Connection strings used to connect to a database are configured on the **Add a DSN** forms in DBMS.

Once successfully added, database connections are displayed in the left pane in the DBMS user interface, along with the tables and SQL statements associated with the connection.



A database connection node represents a DSN or connection string used to connect to a database. A connection is active if an ADO connection can be established using the parameters provided.

After connections have been added, clicking the words **Database connections** in the left pane displays a list of database connections and their connection status (connected or not). You can also click a specific connection.

Note

You can only connect to an existing MySQL database on the MySQL server. The DBMS application cannot be used to create a new database.

This section describes how to add, edit, and delete DSN-based and DSN-less database connections.

In this section:

"DSN-based Database Connections (DBMS)" on page 154

"DSN-less Database Connections (DBMS)" on page 159

DSN-based Database Connections (DBMS)

Data source names (DSNs) are collections of information required for connecting to specific databases. They make it easier to connect to a database because all information can be provided by referencing the DSN rather than providing the entire connection string. The DBMS tool allows you to connect to a database using DSN-based or DSN-less connections. This section discusses the configuration of DSN-based connections.

Note

To use a DSN-based connection, the DSN must reside on the same ASP Server as the DBMS application (for example, the DSN cannot reside on solaris-02 and the DBMS application on solaris-03). Otherwise, a DSN-less connection must be used.

In this section:

"Adding a DSN-based Connection (DBMS)" on page 155

"Editing a DSN-based Connection (DBMS)" on page 157

"Deleting a DSN-based Connection (DBMS)" on page 158

See also:

"DSN-less Database Connections (DBMS)" on page 159

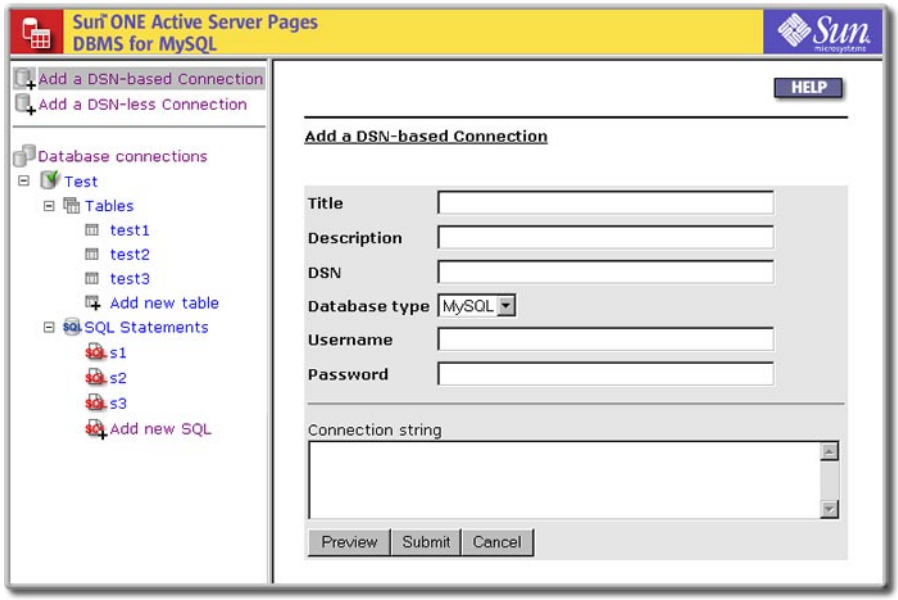
Adding a DSN-based Connection (DBMS)

DSN-based connections can be added by specifying parameters in an HTML form that are then used to construct a connection string, or by entering the entire connection string. Use the following procedure to add a DSN-based connection. Click **Cancel** at any time to cancel the action.

To add a DSN-based connection (DBMS)

1. Open DBMS using the URL provided by your Sun ONE ASP administrator.
2. Click **Add a DSN-based Connection** in the left pane.

   The **Add a DSN-based Connection** form displays.



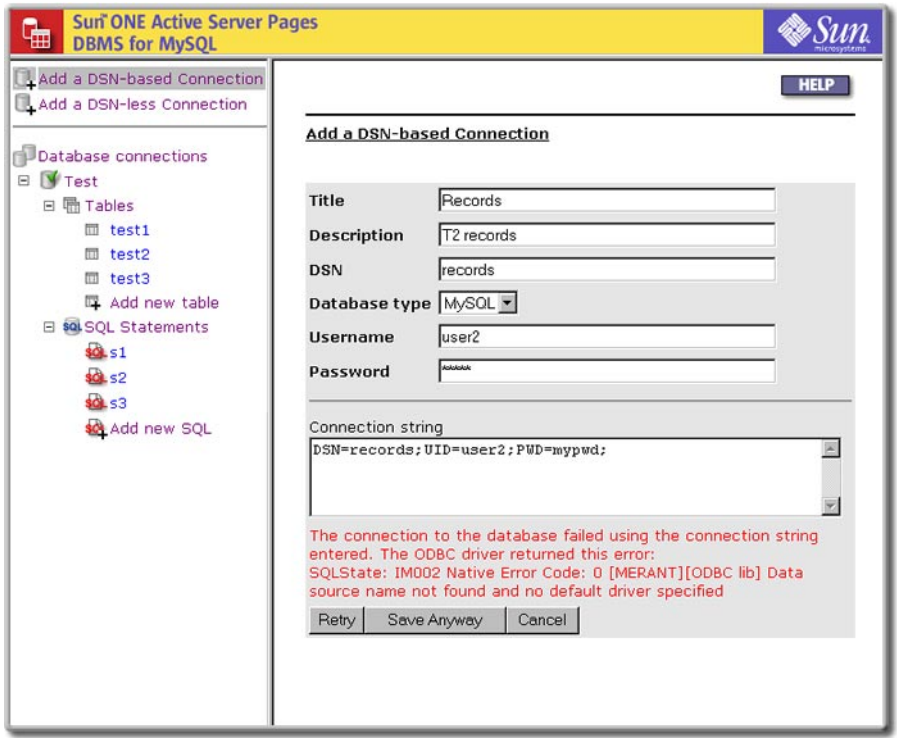
3. In the **Title** box, type the title for the connection string. This title represents the connection string and will be displayed in the DBMS interface.
4. In the **Description** box, type a description of the connection to help distinguish it from others.
5. Provide connection information. Either:
   - Use the form to provide connection information, as described in the remaining steps.

     - or -

   - Enter the entire connection string directly in the **Connection string** box, and then go to step 9.
6. In the **DSN** box, type the name of the DSN.
7. In the **Username** box, type the username to be used for accessing the database.

8. In the **Password** box, type the password to be used for accessing the database.
9. To preview the connection string before using it to connect to the database, click **Preview**.

   A connection string is generated and displayed in the **Connection string** box.
10. To establish a connection using the connection string, click **Submit**. The connection will either succeed or fail:
    - If the string information is correct, a database connection is established, and the connection is displayed in the left pane in the DBMS interface.
    - If the string information is incorrect, the connection will fail, and you will be advised of the error.

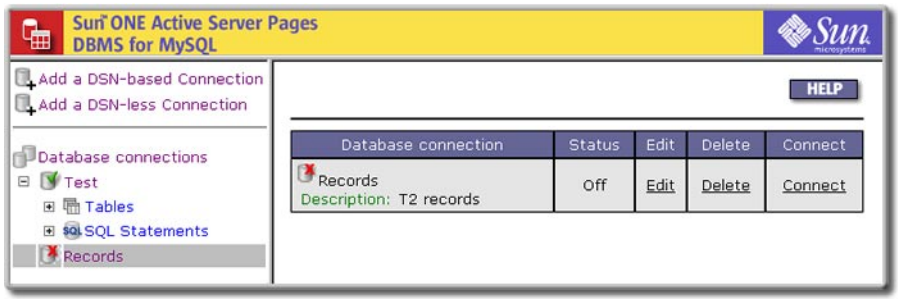


You can do one of the following:

Try to connect again by making changes in the form and then clicking **Retry**.

- or -

Save the connection string anyway by clicking **Save Anyway**. The inactive connection will be saved and displayed in the DBMS interface with a red **X**, indicating a connection cannot be established with the parameters provided.



Changes to the connection string can be made at a later time, as described in "Editing a DSN-based Connection (DBMS)" on page 157.

See also:

"DSN-based Database Connections (DBMS)" on page 154

"Editing a DSN-based Connection (DBMS)" on page 157

"Deleting a DSN-based Connection (DBMS)" on page 158

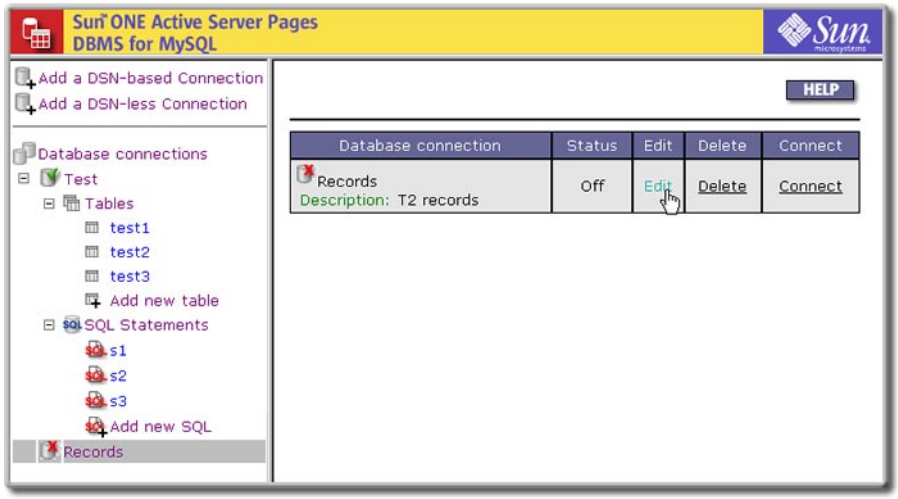
Editing a DSN-based Connection (DBMS)

Use the following procedure to edit a DSN-based connection after it has been created. Click **Cancel** at any time to cancel the action and revert to the settings that were last saved.

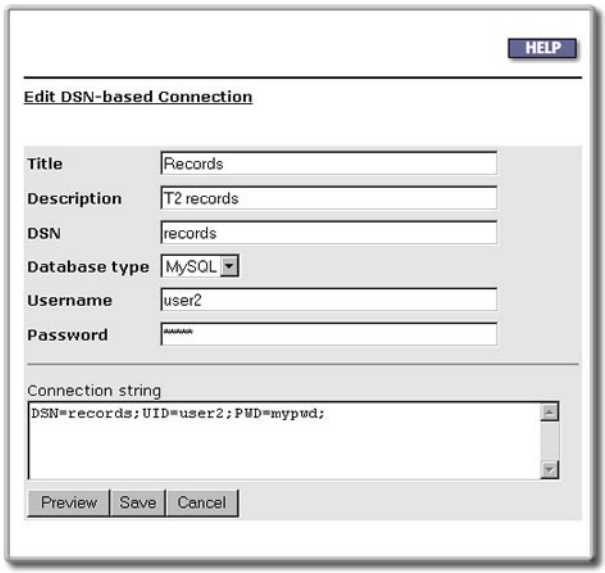
To edit a DSN-based connection (DBMS)

1. Open DBMS using the URL provided by your Sun ONE ASP administrator.
2. Click **Database connections** in the left pane, or click a specific connection.

   Connection information displays in the right pane.
3. In line with the database connection you want to edit, click **Edit**.



The **Edit DSN-based Connection** form displays. (If you are working with an inactive connection, as indicated by a red **X**, you can also click **Connect** to display the edit form.)



4. Change the information as desired, and then click **Preview** to preview the changes, or **Save** to save and submit them. The connection will either succeed or fail:

   - If the changes are valid, a database connection is established, and the edited connection is displayed in the left pane in the DBMS interface.
   - If the changes are invalid, the connection will fail, and you will be advised of the error. You can do one of the following:

     Try to connect again by making changes in the form and then clicking **Retry**.

     - or -

     Save the edited connection string anyway by clicking **Save Anyway**. The inactive connection will be saved and displayed in the DBMS interface with a red **X**, indicating a connection cannot be established with the parameters provided. Changes to the connection string can be made at a later time.

See also:

"DSN-based Database Connections (DBMS)" on page 154

"Adding a DSN-based Connection (DBMS)" on page 155

"Deleting a DSN-based Connection (DBMS)" on page 158

Deleting a DSN-based Connection (DBMS)

Use the following procedure to delete a DSN-based connection. A connection cannot be recovered after it has been deleted.

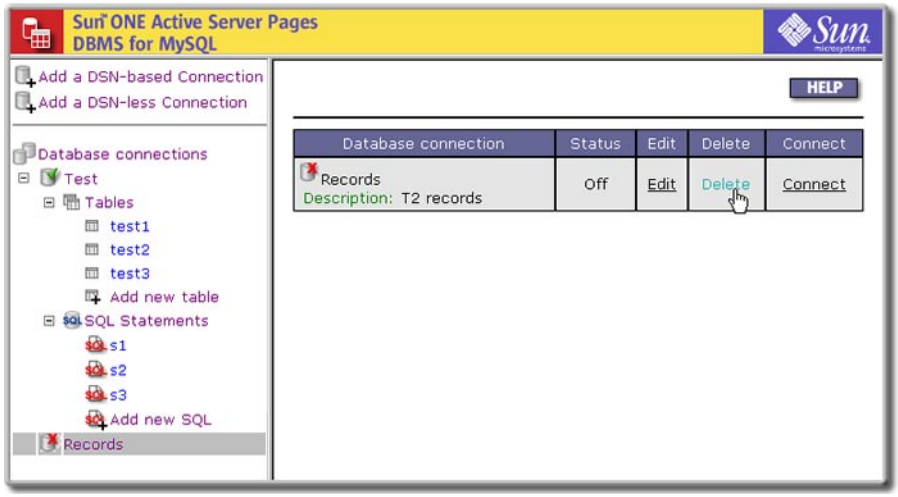
Note

Deleting a connection does not affect the data in your database, but all SQL statements (if any) will be lost.

To delete a DSN-based connection (DBMS)

1. Open DBMS using the URL provided by your Sun ONE ASP administrator.
2. Click **Database connections** in the left pane, or click a specific connection.

   Connection information displays in the right pane.
3. In line with the database connection you want to delete, click **Delete**.



A message displays, asking you to confirm deletion. A connection cannot be recovered after it has been deleted.

See also:

"DSN-based Database Connections (DBMS)" on page 154

"Adding a DSN-based Connection (DBMS)" on page 155

"Editing a DSN-based Connection (DBMS)" on page 157

DSN-less Database Connections (DBMS)

DSN-less connections use connection strings that include all information needed to connect to a database, rather than incorporating the information by referencing a DSN. The DBMS tool allows you to connect to a database using DSN-less and DSN-based connections. This section discusses the configuration of DSN-less connections.

In this section:

"Adding a DSN-less Connection (DBMS)" on page 160

"Editing a DSN-less Connection (DBMS)" on page 162

"Deleting a DSN-less Connection (DBMS)" on page 164

See also:

"DSN-based Database Connections (DBMS)" on page 154

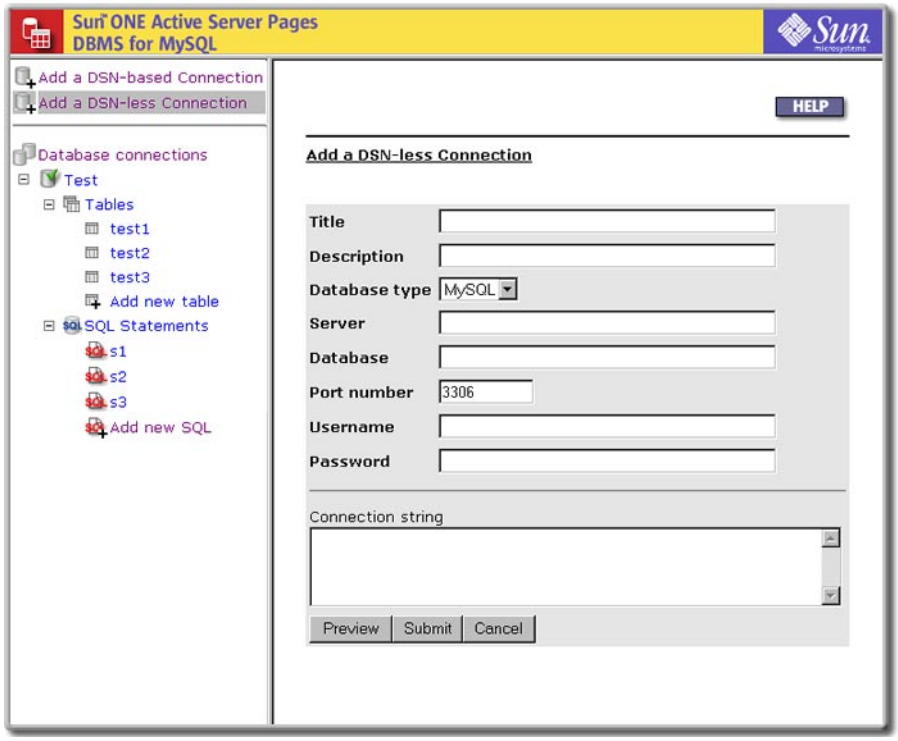
Adding a DSN-less Connection (DBMS)

DSN-less connections can be added by specifying parameters in an HTML form that are then used to construct a connection string, or by entering the entire connection string. Use the following procedure to add a DSN-less connection. Click **Cancel** at any time to cancel the action.

To add a DSN-less connection (DBMS)

1. Open DBMS using the URL provided by your Sun ONE ASP administrator.
2. Click **Add a DSN-less Connection** in the left pane.

   The **Add a DSN-less Connection** form displays.



3. In the **Title** box, type the title for the connection string. This title represents the connection string and will be displayed in the DBMS interface.
4. In the **Description** box, type a description of the connection to help distinguish it from others.
5. Provide connection information. Either:
   - Use the form to provide connection information, as described in the remaining steps.

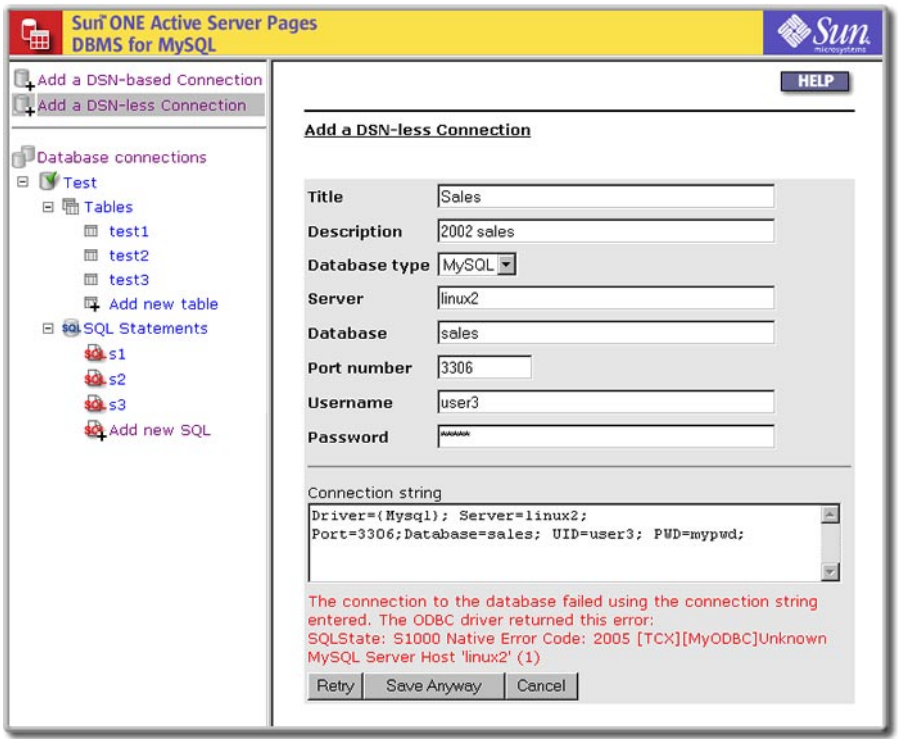
     - or -

- Enter the entire connection string directly in the **Connection string** box, and then go to step 11.

6. In the **Server** box, type the name of the database server.
7. In the **Database** box, type the database name.
8. In the **Port number** box, specify the port on which the database server is configured to listen (3306 by default).
9. In the **Username** box, type the username to be used for accessing the database.
10. In the **Password** box, type the password to be used for accessing the database.
11. To preview the connection string before using it to connect to the database, click **Preview**.

    A connection string is generated and displayed in the **Connection string** box.
12. To establish a connection using the connection string, click **Submit**. The connection will either succeed or fail:

    - If the string information is correct, a database connection is established, and the connection is displayed in the left pane in the DBMS interface.
    - If the string information is incorrect, the connection will fail, and you will be advised of the error.

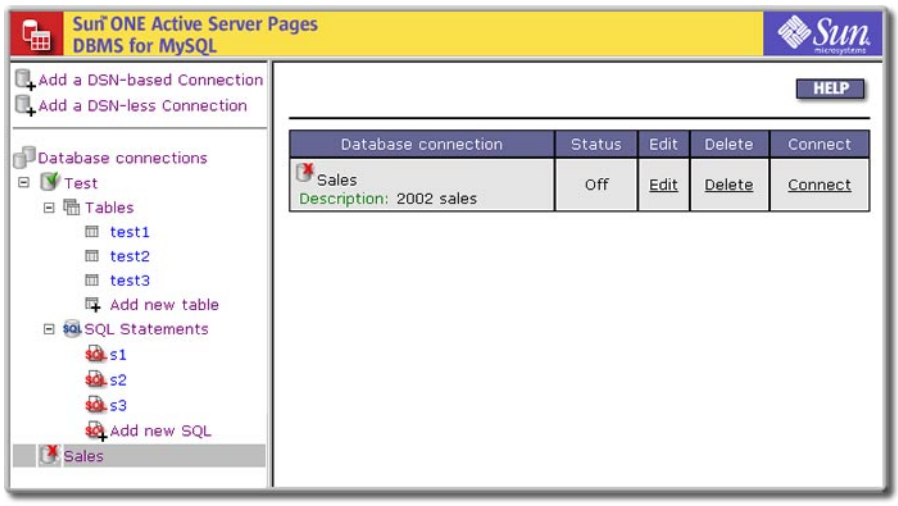


You can do one of the following:

Try to connect again by making changes in the form and then clicking **Retry**.

- or -

Save the connection string anyway by clicking **Save Anyway**. The inactive connection will be saved and displayed in the DBMS interface with a red **X**, indicating a connection cannot be established with the parameters provided.



Changes to the connection string can be made at a later time, as described in "Editing a DSN-based Connection (DBMS)" on page 157.

See also:

"DSN-less Database Connections (DBMS)" on page 159

"Editing a DSN-less Connection (DBMS)" on page 162

"Deleting a DSN-less Connection (DBMS)" on page 164

Editing a DSN-less Connection (DBMS)

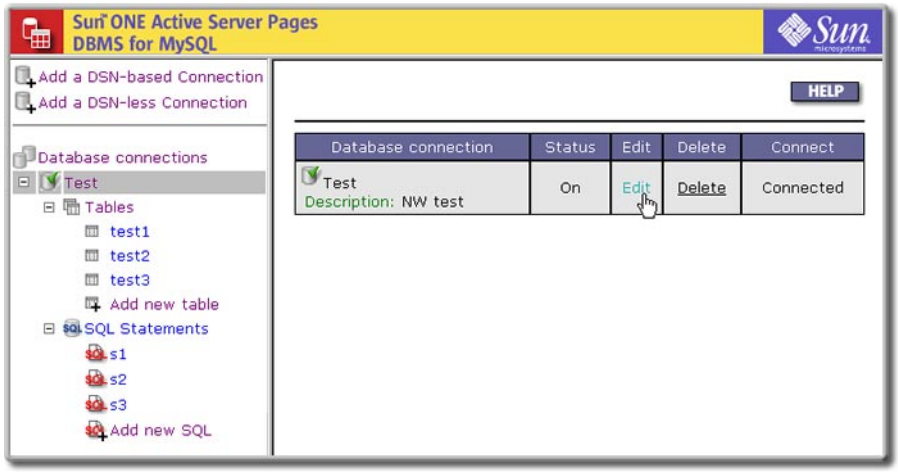
Use the following procedure to edit a DSN-less connection after it has been created. Click **Cancel** at any time to cancel the action and revert to the settings that were last saved.

To edit a DSN-less connection (DBMS)

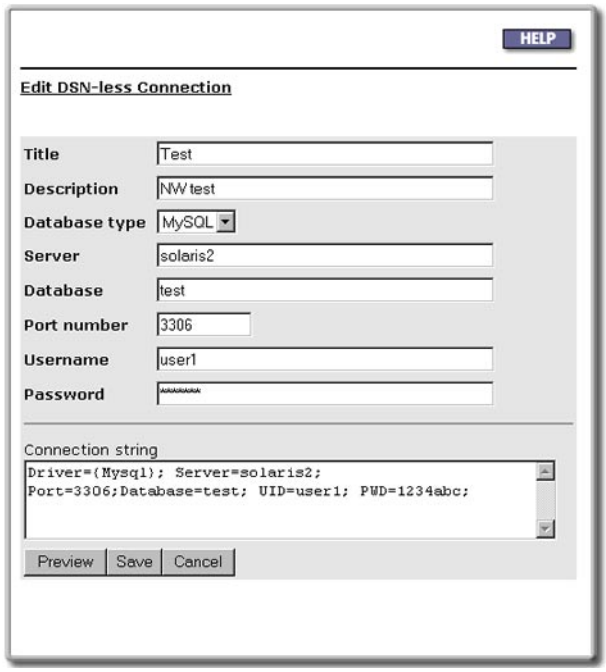
1. Open DBMS using the URL provided by your Sun ONE ASP administrator.
2. Click **Database connections** in the left pane, or click a specific connection.

   Connection information displays in the right pane.

3. In line with the database connection you want to edit, click **Edit**.



The **Edit DSN-less Connection** form displays. (If you are working with an inactive connection, as indicated by a red **X**, you can also click **Connect** to display the edit form.)



4. Change the information as desired, and then click **Preview** to preview the changes, or **Save** to save and submit them. The connection will either succeed or fail:

   - If the changes are valid, a database connection is established, and the edited connection is displayed in the left pane in the DBMS interface.
   - If the changes are invalid, the connection will fail, and you will be advised of the error. You can do one of the following:

Try to connect again by making changes in the form and then clicking **Retry**.

- or -

Save the edited connection string anyway by clicking **Save Anyway**. The inactive connection will be saved and displayed in the DBMS interface with a red **X**, indicating a connection cannot be established with the parameters provided. Changes to the connection string can be made at a later time.

See also:

"DSN-less Database Connections (DBMS)" on page 159

"Adding a DSN-less Connection (DBMS)" on page 160

"Deleting a DSN-less Connection (DBMS)" on page 164

Deleting a DSN-less Connection (DBMS)

Use the following procedure to delete a DSN-less connection. A connection cannot be recovered after it has been deleted.

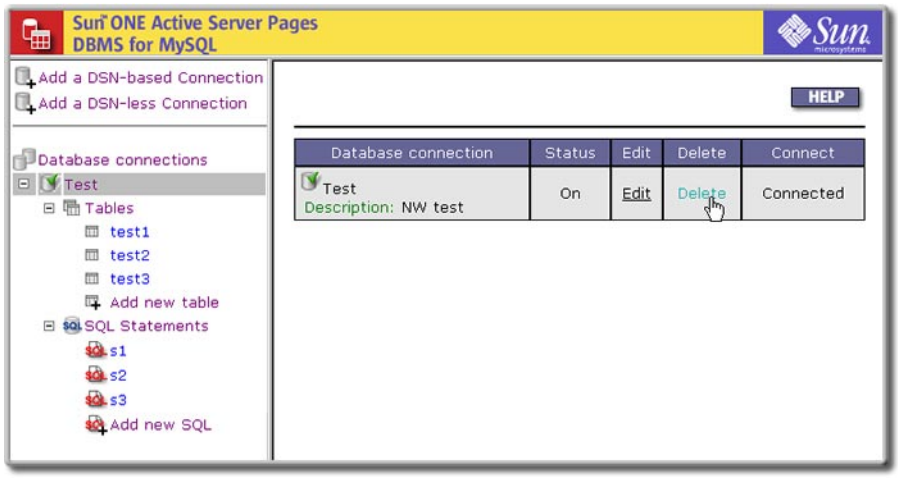
Note

Deleting a connection does not affect the data in your database, but all SQL statements (if any) will be lost.

To delete a DSN-less connection (DBMS)

1. Open DBMS using the URL provided by your Sun ONE ASP administrator.
2. Click **Database connections** in the left pane, or click a specific connection.

   Connection information displays in the right pane.
3. In line with the database connection you want to delete, click **Delete**.



A message displays, asking you to confirm deletion. A connection cannot be recovered after it has been deleted.

See also:

"DSN-less Database Connections (DBMS)" on page 159

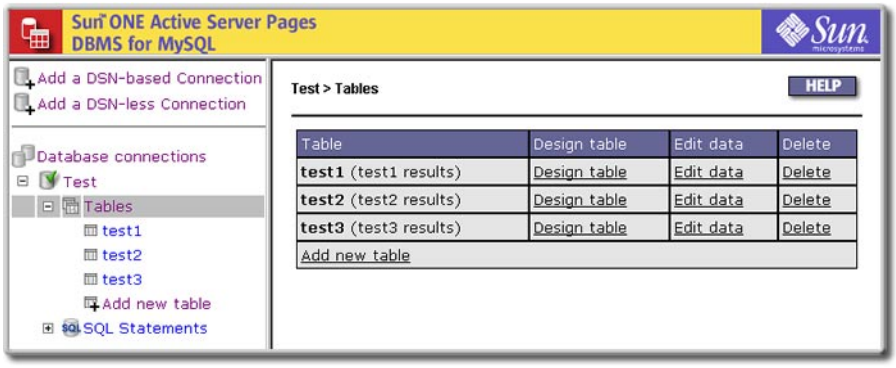
"Adding a DSN-less Connection (DBMS)" on page 160

"Editing a DSN-less Connection (DBMS)" on page 162

Working with Tables

The DBMS application can be used to perform data maintenance and query a database using SQL statements. Users with full access to the MySQL database can add, update, and delete tables.

Table names are displayed in the left pane of the DBMS interface, and corresponding properties are displayed in the right pane. Clicking the word **Tables** in the interface displays a list of tables that have been created and saved.



You can also click the plus (+) sign to expand the list of tables. Clicking a specific table displays that table. This section describes how to add, update, and delete tables.

Note

The tables displayed and the actions you can take in DBMS are governed solely by the privileges granted to you by the MySQL database administrator.

In this section:

"Adding New Tables" on page 166

"Updating Existing Tables" on page 168

"Deleting Tables" on page 173

Data Validation

Server-side data validation is performed when adding new records or updating existing tables. That validation includes NULL checking and numeric validation:

- For NULL validation, an error is reported if a column that is not nullable is left empty.
- For numeric fields, an error is reported if inputs are not numeric.

- For other errors, an ODBC error description is returned.

Adding New Tables

New tables require a table name and at least one table column. Names and column types must adhere to MySQL conventions (see "DBMS Conventions" on page 152). Use the following procedure to add a new table.

Note

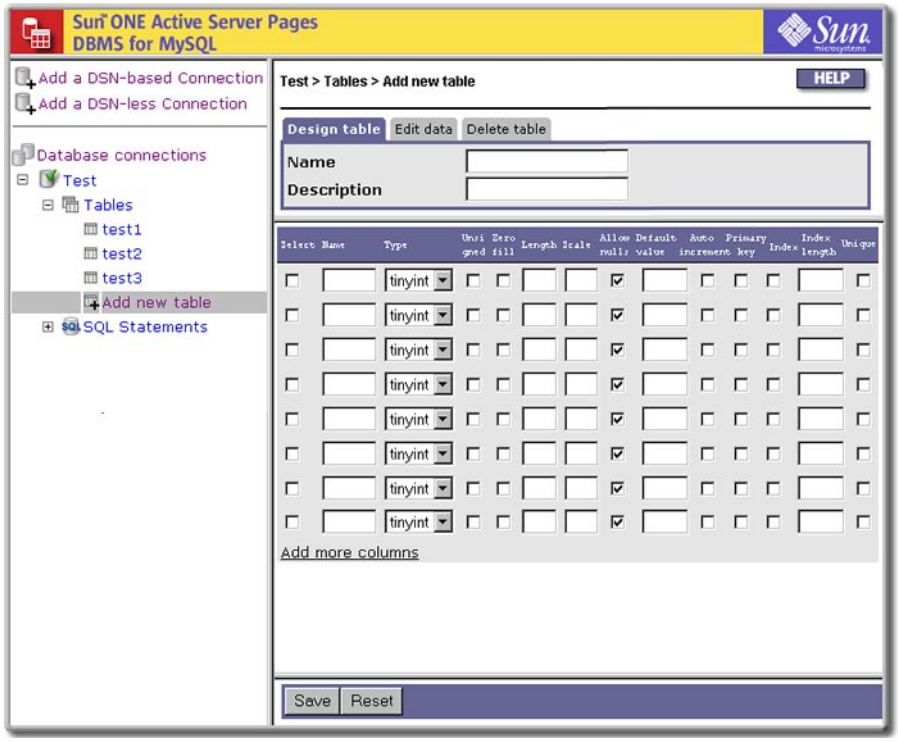
You cannot perform this procedure unless corresponding privileges have been granted by the MySQL database administrator.

To add a new table

1. Open DBMS using the URL provided by your Sun ONE ASP administrator.
2. Click a database connection in the left pane to expand a node if necessary, and then expand the **Tables** node.

   Table information displays in the right pane.
3. Click **Add new table** in the left pane, or in the table in the right pane.

   The **Add new table** page displays, opening on the **Design table** tab by default. This page is used to design the table.



4. In the **Name** box for the new table, type the table name.
5. In the **Description** box, type a table description.
6. In the **Name** box for a row, type the column name.
7. In the remaining fields, specify column information as desired. It is strongly recommended that a primary key be added to the table. Also note the following:
   - The **Select** box for each row is selected automatically as you work with the column rows. Selected rows will be included in the table.
   - To add more columns to the table, click **Add more columns** at the bottom of the **Add new table** page. Ten blank rows of input boxes will be added.
   - Default settings based on data type are provided for some fields. For descriptions of each field and detailed information about MySQL conventions and rules, consult MySQL documentation at the following URL (particularly the language reference section):

     http://www.mysql.com/documentation
8. After designing the new table, do one of the following:
   - Click **Save** to submit the information for the rows that are selected and save the table (if the table cannot be saved, an error message displays).

     - or -
   - Click **Reset** to clear the form.

Note

After a new table has been saved you can add, edit, or remove data as described in "Updating Existing Tables" on page 168 (assuming corresponding privileges have been granted by the database administrator). You can also change table design as described in "Changing Table Design" on page 168.

See also:

"Working with SQL Statements" on page 174

Updating Existing Tables

Users with full access to the MySQL database can change table design, add and edit data, and delete the table. You cannot perform the procedures in this section unless corresponding privileges have been granted by the MySQL database administrator.

Note

TIME values in Sun ONE ASP DBMS are displayed in HH:MM:SS format ONLY. The -HHH:MM:SS to HHH:MM:SS formats supported in MySQL are not supported in DBMS. Therefore, if you are viewing a MySQL database that uses these unsupported TIME formats, the time will not display correctly in the DBMS tool. Likewise, TIME values cannot be entered in unsupported formats; HH:MM:SS must be used.

In this section:

Changing Table Design

Use the following procedure to change table design for an existing table.

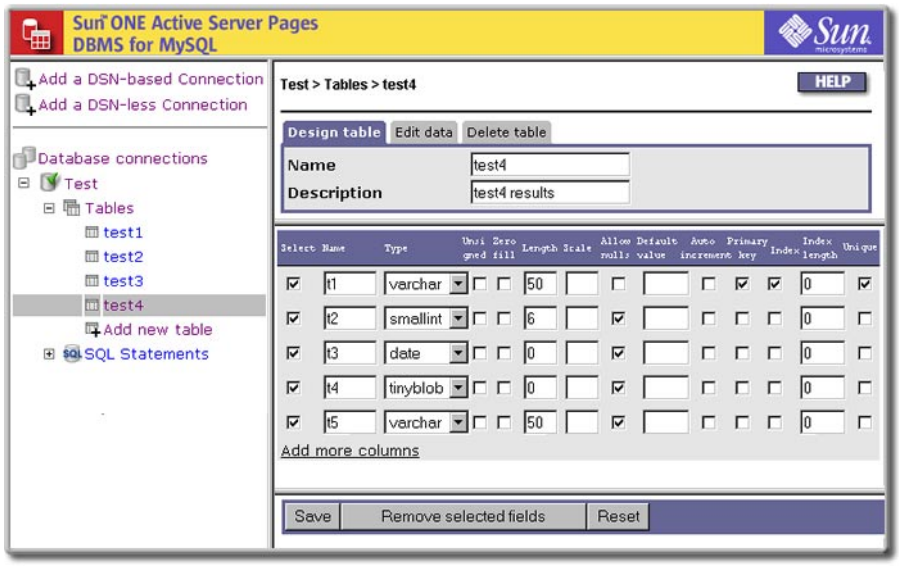
To change table design

1. Open DBMS using the URL provided by your Sun ONE ASP administrator.
2. Do one of the following:
   - Expand the **Tables** node, click a specific table in the left pane, and then click the **Design table** tab in the right pane.

     - or -

   - Click the word **Tables** under a database connection in the left pane, and then click **Design table** in the right pane in line with the table you want to change. The table displays on the **Design table** tab by default.

   The **Add new table** page displays, opening on the **Design table** tab by default. This page is used to design the table.



3. Make changes as desired (columns with changes will be selected by default). Note the following:

   - To add more columns to the table, click **Add more columns** at the bottom of the page. Ten blank rows of input boxes will be added.
   - Default settings based on data type are provided for some fields. For descriptions of each field and detailed information about MySQL conventions and rules, consult MySQL documentation at the following URL (particularly the language reference section):

     http://www.mysql.com/documentation

4. Click **Save** to submit the selected information and save the table, or **Reset** to revert to the data that was last saved.

Note

To remove entire columns, select the columns and click **Remove selected fields**.

Adding Data

Use the following procedures to add data (records) to an existing table.

Note

TIME values in Sun ONE ASP DBMS are displayed in HH:MM:SS format ONLY. The -HHH:MM:SS to HHH:MM:SS formats supported in MySQL are not supported in DBMS. Therefore, if you are viewing a MySQL database that uses these unsupported TIME formats, the time will not display correctly in the DBMS tool. Likewise, TIME values cannot be entered in unsupported formats; HH:MM:SS must be used.

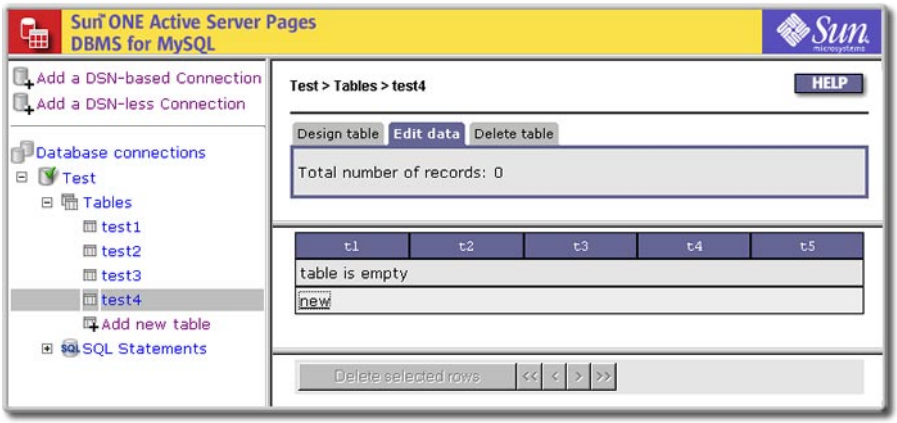
To add data

1. Open DBMS using the URL provided by your Sun ONE ASP administrator.
2. Do one of the following:
   - Expand the **Tables** node and click a specific table.

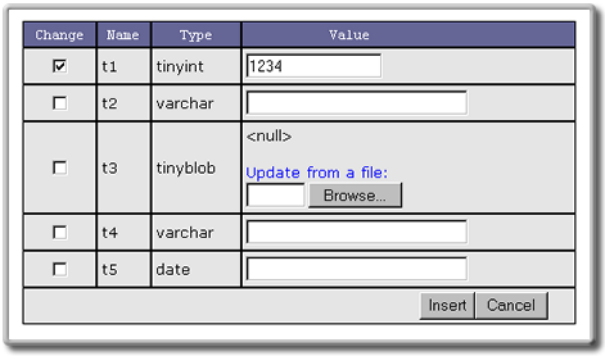
     - or -

   - Click the word **Tables** under a database connection, and then click **Edit data** in the right pane in line with the table you want to change.

   The table opens on the **Edit data** tab by default.



3. Click **new** in the right pane to add a new record.
4. In the form that displays, add record data. As you add data, columns will be selected by default (the box in the **Change** column will be selected).



5. Click **Insert** to submit the selected information and add the new data.

   The new data displays on the **Edit data** tab and your changes will be saved (if the new data is not visible, you may need to refresh your browser by right-clicking in the pane and then clicking **Refresh**).

Note

The **Edit data** tab also provides a navigation bar used to navigate records. When a table is open, the first 20 rows of records are displayed by default. To change the number of records displayed, change the number in the **#Shown**

box, and then click **Refresh**. If the total number of records exceeds the number shown, use the navigation buttons (< > symbols) in the lower right corner to navigate through the records. Clicking << or >> takes you to the first or last set of records. Records will be displayed in increments of the number in the **#Shown** box.

See also:

"Changing Table Design" on page 168

Editing Data

Use the following procedure to edit data (records) in an existing table.

Note

TIME values in Sun ONE ASP DBMS are displayed in HH:MM:SS format ONLY. The -HHH:MM:SS to HHH:MM:SS formats supported in MySQL are not supported in DBMS. Therefore, if you are viewing a MySQL database that uses these unsupported TIME formats, the time will not display correctly in the DBMS tool. Likewise, TIME values cannot be entered in unsupported formats; HH:MM:SS must be used.

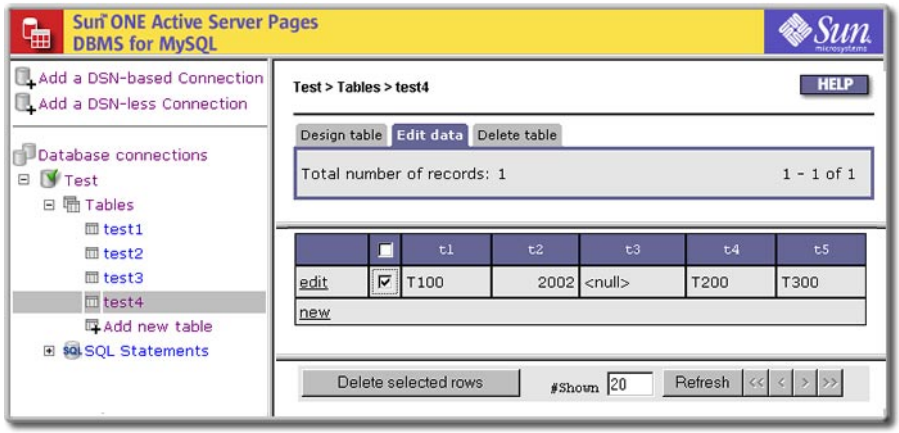
To edit data

1. Open DBMS using the URL provided by your Sun ONE ASP administrator.
2. Do one of the following:
   - Expand the **Tables** node and click a specific table.

     - or -

   - Click the word **Tables** under a database connection in the left pane, and then click **Edit data** in the right pane in line with the table you want to edit.

   The table opens on the **Edit data** tab by default, displaying record rows. Each record row has an **edit** link.



3. In line with the record you want to edit, click **edit**.
4. In the form that displays, make changes as desired and then click **Update** to submit the selected changes (records with changes will be selected by default).

   The edited data displays on the **Edit data** tab and your changes will be saved (if the changes are not visible, you may need to refresh your browser by right-clicking in the pane and then clicking **Refresh**).

Note

The **Edit data** tab also provides a navigation bar used to navigate records. When a table is open, the first 20 rows of records are displayed by default. To change the number of records displayed, change the number in the **#Shown** box, and then click **Refresh**. If the total number of records exceeds the number shown, use the navigation buttons (< > symbols) in the lower right corner to navigate through the records. Clicking << or >> takes you to the first or last set of records. Records are displayed in increments of the number in the **#Shown** box.

Deleting Data

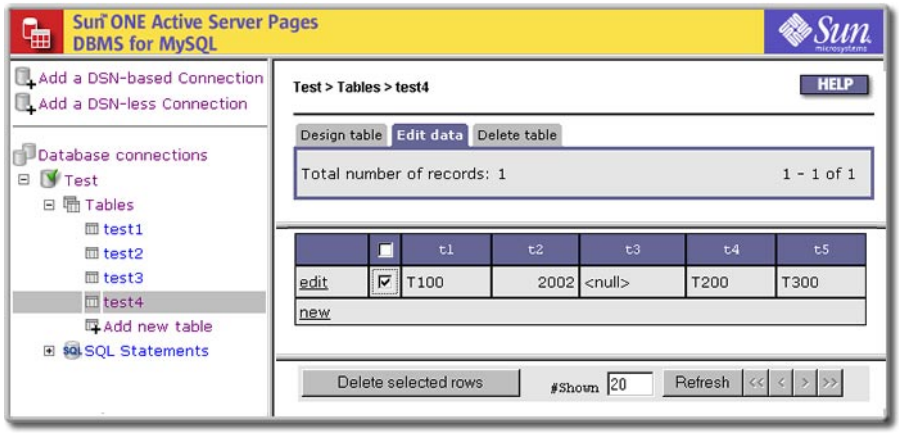
Use the following procedure to delete data (records) from an existing table. A record cannot be recovered after it has been deleted.

1. Open DBMS using the URL provided by your Sun ONE ASP administrator.
2. Do one of the following:
   - Expand the **Tables** node and click a specific table.

     - or -

   - Click the word **Tables** under a database connection in the left pane, and then click **Edit data** in the right pane in line with the table you want to change.

   The table opens on the **Edit data** tab by default.



3. Select a specific row or rows, and then click **Delete selected rows**. To select all rows, select the check box in the very first row (the box in the title bar not associated with a record).

   A message displays, asking you to confirm deletion. A record cannot be recovered after it has been deleted.

Note

The **Edit data** tab also provides a navigation bar used to navigate records. When a table is open, the first 20 rows of records are displayed by default. To change the number of records displayed, change the number in the **#Shown** box, and then click **Refresh**. If the total number of records exceeds the number shown, use the navigation buttons (< > symbols) in the lower right corner to navigate through the records. Clicking << or >> takes you to the first or last set of records. Records are displayed in increments of the number in the **#Shown** box.

See also:

"Changing Table Design" on page 168

Deleting Tables

Use the following procedure to delete a table. A table cannot be recovered after it has been deleted.

To delete a table

1. Open DBMS using the URL provided by your Sun ONE ASP administrator.
2. Do one of the following:
   - Expand the **Tables** node, click a specific table, and then click the **Delete table** tab in the right pane.

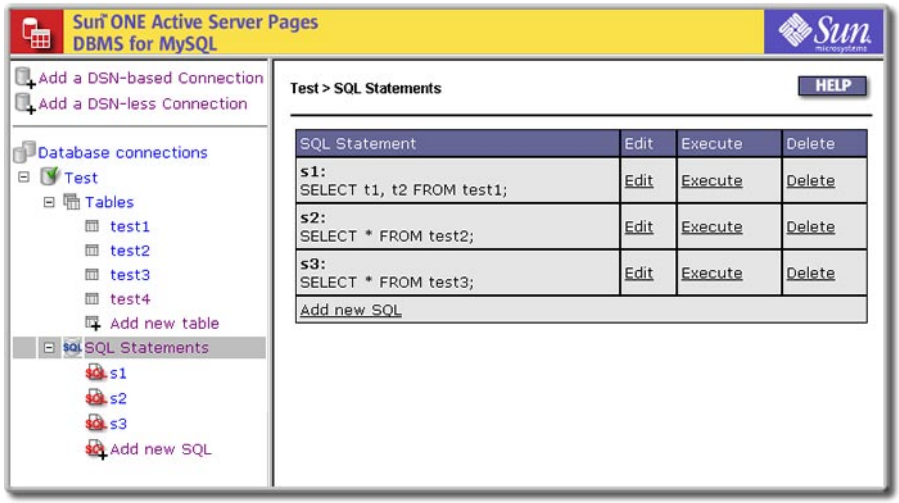
     - or -

   - Click the word **Tables** under a database connection in the left pane, and then click **Delete** in the right pane in line with the table you want to delete.

   In both cases a message displays, asking you to confirm deletion. A table cannot be recovered after it has been deleted.

Working with SQL Statements

The DBMS interface can be used to query a database using SQL statements. SQL statements are displayed in the left pane of the DBMS interface, and corresponding properties are displayed in the right pane. Clicking the words **SQL Statements** in the interface displays a list of SQL statements that have been created and saved. You can also click the plus (+) sign to expand the list of SQL statements. Clicking a specific SQL statement displays that statement.



SQL statements are computer specific, which means that SQL statements created when using DBMS on one computer will not be visible when DBMS is accessed from another computer. You can add, save, and execute the most commonly used SQL statements (SQL statements are saved in DBMS, not in your database). This section describes how to add, edit, execute, and delete SQL statements.

Note

The actions you can take are governed solely by the privileges granted to you by the MySQL database administrator.

In this section:

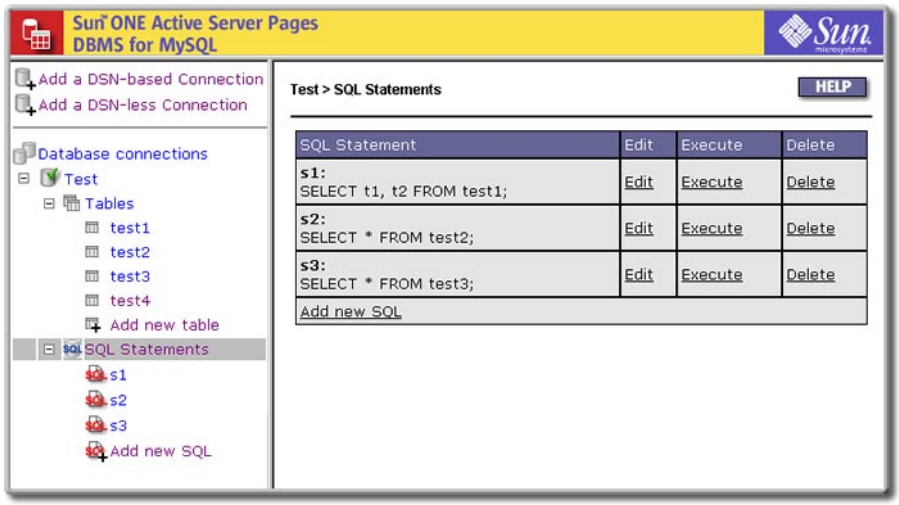
Adding SQL Statements

Use the following procedure to add a new SQL statement. (You can also create a new SQL statement by saving an existing statement with a new name. For more information, see "Editing SQL Statements" on page 176.)

To add a new SQL statement

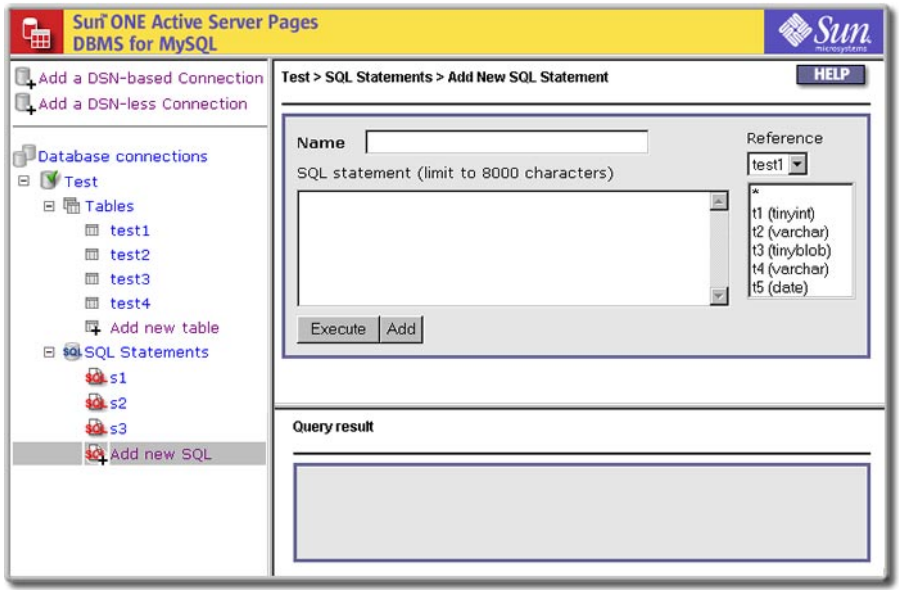
1. Open DBMS using the URL provided by your Sun ONE ASP administrator.
2. Click a database connection node in the left pane to expand the node if necessary, and then expand the **SQL Statements** node.

   SQL statement information displays in the right pane.



3. Click **Add new SQL** in the left pane, or in the table in the right pane.

   The **Add New SQL Statement** page displays.



   The page is divided into two panes. The top pane contains a form for entering the SQL statement. The lower pane displays a message box where query results and execution or error messages are displayed.

4. In the **Name** box, type the name of the SQL statement.

5. Do one of the following:
   - Type the SQL statement in the **SQL statement** box. Only SQL statements permitted in MySQL can be entered, such as a conventional SELECT statement, or statements specific to MySQL such as DESCRIBE <*Table Name*>.

     - or -
   - Use the **Reference** boxes to create a SELECT statement or view the data structure of a particular table (the top reference box displays tables, the lower box displays columns).
6. Do one of the following:
   - Click **Execute** to run the query and return results to the **Query result** box in the right pane. Results are returned as either a recordset or a message. If the query returns a recordset, the result is listed by pages. Otherwise, a message of some type displays. If the query is unsuccessful, an error message displays.

     - or -
   - Click **Add** to add and save the new statement. If the SQL statement is successfully added, a message to that effect displays. Click **OK**, and the new SQL statement appears in the DBMS interface.

Note

The **#shown** box displayed with query results specifies how many records should be displayed. This number is set to 20 by default. To see a different number of records, change the number in the box and then click **Refresh**. If the total number of records exceeds the number shown, use the navigation buttons (**<** **>** symbols) in the lower right corner to navigate through the records. Clicking **<<** or **>>** takes you to the first or last set of records. Records are displayed in increments of the number in the **#shown** box.

See also:

"Working with SQL Statements" on page 174

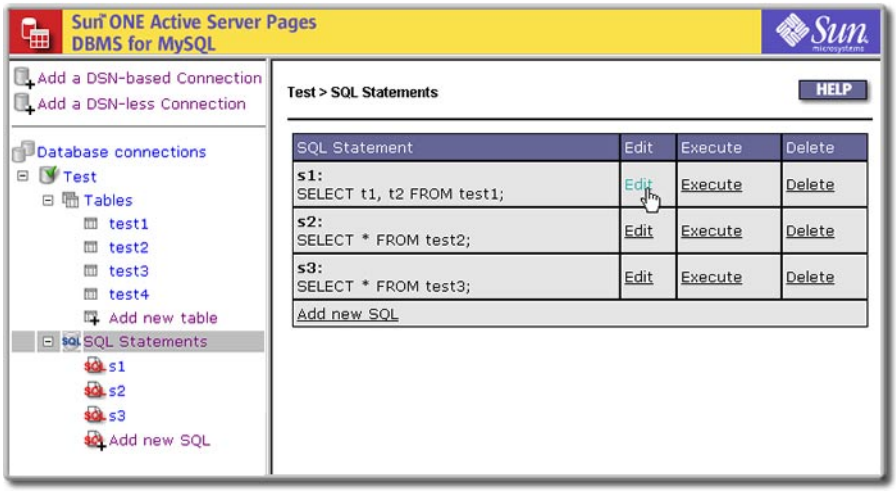
Editing SQL Statements

Use the following procedure to edit existing SQL statements.

To edit SQL statements

1. Open DBMS using the URL provided by your Sun ONE ASP administrator.
2. Under a connection node in the left pane, do one of the following:
   - Expand the **SQL Statements** node and click a specific statement.

     - or -
   - Click the words **SQL Statements**, and then click **Edit** in line with the SQL statement you want to edit.



3. Edit the SQL statement directly in the SQL statement box or by using the **Reference** boxes, and then take the desired action:

   - Click **Execute** to run the query and return results to the **Query result** box in the right pane. Results are returned as either a recordset or a message. If the query returns a recordset, the result is listed by pages. Otherwise, a message of some type displays. If the query is unsuccessful, an error message displays.
   - Click **Save** to save the SQL statement.
   - Enter a new name in the box to the right of the **Save As** button, and then click **Save As** to save the SQL statement with a new name.

Note

The **#shown** box displayed with query results specifies how many records should be displayed. This number is set to 20 by default. To see a different number of records, change the number in the box and then click **Refresh**. If the total number of records exceeds the number shown, use the navigation buttons (< > symbols) in the lower right corner to navigate through the records. Clicking << or >> takes you to the first or last set of records. Records are displayed in increments of the number in the **#shown** box.

See also:

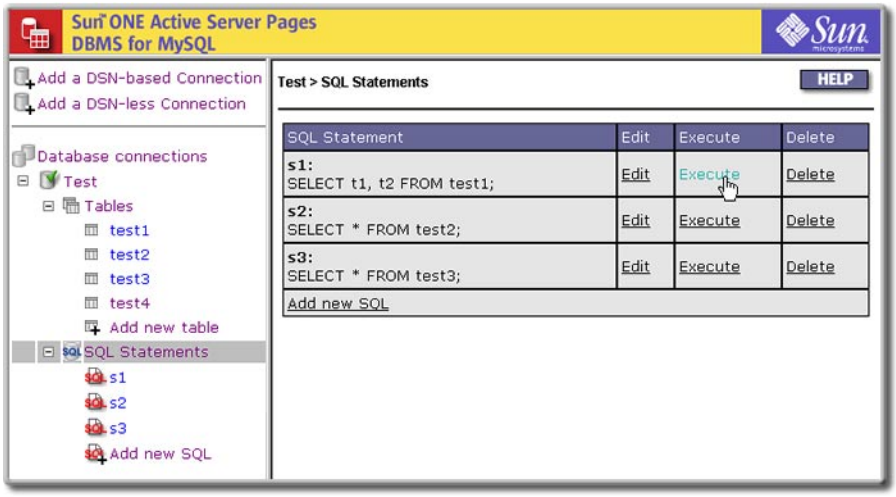
"Working with SQL Statements" on page 174

Executing SQL Statements

Use the following procedure to execute SQL statements.

To execute SQL statements

1. Open DBMS using the URL provided by your Sun ONE ASP administrator.
2. Do one of the following:
   - To execute an existing SQL statement, click the words **SQL Statements** in the left pane, and then click **Execute** in the right pane in line with the SQL statement you want to execute.



   You can also click a specific SQL statement in the left pane, and then click **Execute** in the right pane.

   - or -

   - To add and execute a new SQL statement at the same time, perform the procedure described in "Adding SQL Statements" on page 174.

After clicking **Execute** to run the query, results are returned to the **Query result** box in the right pane as either a recordset or a message.

If the query returns a recordset, the result is listed by pages. Otherwise, a message of some type displays. If the query is unsuccessful, an error message displays.

Note

The **#shown** box displayed with query results specifies how many records should be displayed. This number is set to 20 by default. To see a different number of records, change the number in the box and then click **Refresh**. If the total number of records exceeds the number shown, use the navigation buttons (< > symbols) in the lower right corner to navigate through the records. Clicking << or >>

takes you to the first or last set of records. Records are displayed in increments of the number in the **#shown** box.

See also:

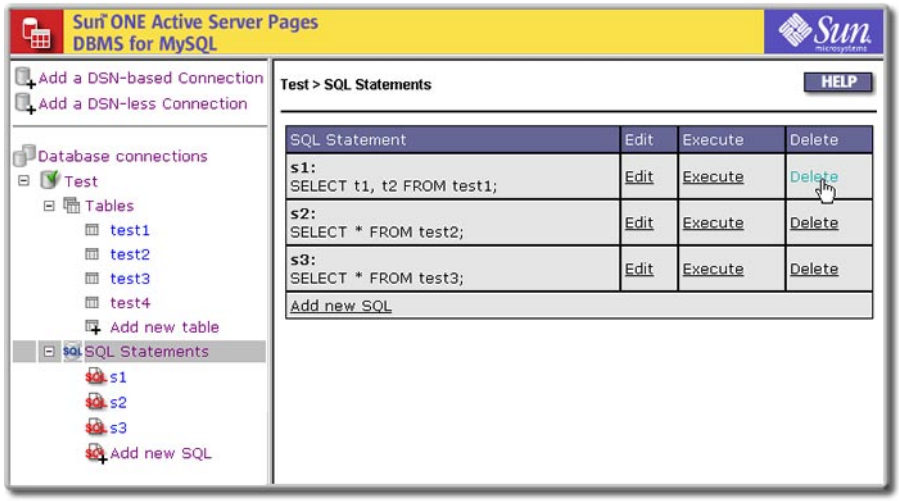
"Working with SQL Statements" on page 174

Deleting SQL Statements

Use the following procedure to delete SQL statements. A SQL statement cannot be recovered after it has been deleted.

To delete SQL statements

1. Open DBMS using the URL provided by your Sun ONE ASP administrator.
2. Under the desired database connection node, click the words **SQL Statements**.
3. In line with the SQL statement you want to delete, click **Delete**.



A message displays, asking you to confirm deletion. A SQL statement cannot be recovered after it has been deleted.

8 Building Sun ONE ASP Applications

ASP enables developers to easily create dynamic Web applications 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 ONE Active Server Pages enables the scripting logic to interface with built-in ASP objects, which automatically handle many menial tasks, making application development easier. In addition to using these basic elements, ASP can be extended by using the Component Object Model (COM), which 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 ONE ASP includes an ADO component that provides a high-performance interface between Web pages and databases that adhere to the ODBC standard. In addition, Sun ONE ASP Chili!Beans support included with Sun ONE ASP enables you to use Java objects with your ASP applications.

This chapter describes how to build a Sun ONE ASP application. The chapter:

- Details the basics of building a Sun ONE ASP application, which involves such steps as creating an ASP page, adding server-side scripts and server-side includes, and defining the application.
- Describes how to extend applications by using objects and components, connecting to databases, and developing applications to publish in locales other than the United States.
- Provides information about publishing a Sun ONE ASP application.

Note

This chapter provides basic information about ASP applications. You may wish to consult additional print and Web resources for more detailed information about developing ASP applications.

In this chapter:

Creating the Basic ASP Application

Sun ONE Active Server Pages combines ASP technology with server-side object-oriented components to provide an integrated Web development environment. A Sun ONE ASP application consists of the following elements:

- ASP files (or pages), which are plain text files with an .asp extension. ASP pages are a combination of standard HTML and scripting, written in either VBScript or JScript.
- Optional components written in a variety of languages. On Windows, Sun ONE ASP server components can be written in any language that supports COM, including Java, Visual Basic, and C++. On UNIX, Sun ONE ASP server components can be written in Java.
- 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 ONE 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 details the basics of building a Sun ONE ASP application.

In this section:

See also:

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 of 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 or Adobe GoLive.

Sun ONE Active Server Pages, combined with one or more of these development tools, provides a common Web application platform for:

- Applications that are large and small, simple or complex.
- Different Web servers and operating systems.
- Developers and page creators with widely varying skill levels.

Note

Sun ONE ASP enables you to run ASP pages generated by a variety of development tools. However, questions about the installation, configuration, and use of a specific tool 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 plain text files with an .asp file name 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 Web developers to easily transform a static Web site into a dynamic one by adding ASP scripts to existing HTML page. With Sun ONE Active Server Pages, you can write scripts in VBScript or JScript. Saving the page with an *.asp file name extension tells the Web server how to process the script commands.

See also:

"Creating the Basic ASP Application" on page 182

"Adding Scripts" on page 184

"Using Server-side Includes" on page 188

"Using @Directives" on page 186

Adding Scripts

Once you have created an ASP page, as described in "Creating an ASP Page" on page 183, 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 ONE 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" on page 185.

As the ASP Server processes each ASP script block, it creates HTML text that is returned to the Web server for rendering. Unlike with client-side scripts, with server-side ASP scripts you do not need to worry 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="Adobe GoLive">

<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" on page 182

"Creating an ASP Page" on page 183

"Using Server-side Includes" on page 188

"Using @Directives" on page 186

Changing the Scripting Language

Sun ONE Active Server Pages provides script interpreters to process the commands in an ASP script. Sun ONE ASP includes the Sun scripting engines, Sun ONE ASP VBScript and Sun ONE ASP JavaScript, which provide functionality equivalent to version 5.5 of Microsoft VBScript and JScript (see "Chapter 15, Scripting Languages Reference" on page 503 for more information about the scripting engines).

The default scripting language is VBScript, but you can change this to JavaScript for an ASP page by using the **@LANGUAGE** directive at the beginning of your ASP file, as described in "Using @Directives" on page 186.

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=JavaScript runat=server>
```

Alternatively, your system administrator can change the default scripting language to either VBScript or JavaScript on the ASP Server. For Windows systems, see "Editing the Windows Registry" on page 515. For UNIX and Linux systems, see "Editing the Sun ONE ASP Configuration File" on page 517.

See also:

"Adding Scripts" on page 184

Using @Directives

Sun ONE Active Server Pages provides @ processing directives, in addition to the available scripting commands. These directives provide information to the Web server about how an .asp file should be processed. The directives listed below are implemented in Sun ONE ASP (the @TRANSACTION directive is not implemented).

- @CODEPAGE
- @ENABLESESSIONSTATE
- @LANGUAGE
- @LCID

Note

Regarding syntax, 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 (=).

See also:

"Creating the Basic ASP Application" on page 182

"Creating an ASP Page" on page 183

"Using Server-side Includes" on page 188

@CODEPAGE Directive

This directive specifies how literal (static) strings are encoded in a Web page. Code pages are character sets and are important in international applications. Code pages are not the same for all languages. Setting the @CODEPAGE directive explicitly affects literal strings in a single response.

The syntax for this directive is as follows:

```
<%@ CODEPAGE=codepage %>
```

The parameters are as follows:

codepage

An integer that represents the character formatting code page. For a list of the code page integers for the languages supported by Sun ONE ASP, see "Configuring International Support" on page 43.

Take note of the following:

- `@CODEPAGE` affects literal (static) strings in a single response.
- **Response.CodePage** affects dynamic strings in a single response.
- **Session.CodePage** affects dynamic strings in all responses in a session.

See also:

"Developing International Applications" on page 212

"Configuring International Support" on page 43

"ASP Session Object CodePage Property" on page 264

@ENABLESESSIONSTATE Directive

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 ONE ASP to process the script. By default, sessions are enabled. For more information, see "Managing User Sessions" on page 192.

The syntax for this directive is as follows:

```
<%@ ENABLESESSIONSTATE=True|False %>
```

@LANGUAGE Directive

By default, the primary scripting language for Sun ONE ASP is Sun ONE ASP VBScript, but this can be changed to Sun ONE ASP JavaScript for each ASP page by using the @LANGUAGE directive at the beginning of the ASP file.

The syntax for this directive is as follows:

```
<%@ LANGUAGE=scriptengine %>
```

The parameters are as follows:

scriptengine

The script engine that should process the script (VBScript or JavaScript).

See also:

"Chapter 15, Scripting Languages Reference" on page 503

@LCID Directive

This directive specifies how dates, times, and currencies are formatted for different locales. A Local Language Identifier (LCID) is a 32-bit value that identifies a geographic locale.

The syntax for this directive is as follows:

```
<%@ LCID=localeidentifier %>
```

The parameters are as follows:

localeidentifier

A valid Locale Indentifier (LCID) number. For a list of valid values, see "Configuring International Support" on page 43.

Take note of the following:

- `@LCID` affects literal (static) strings in a single response.
- **Response.LCID** affects dynamic strings in a single response.
- **Session.LCID** affects dynamic strings in all responses in a session.

See also:

"Developing International Applications" on page 212

"Configuring International Support" on page 43

"ASP Session Object LCID Property" on page 265

Using Server-side Includes

A server-side include directive is used 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"-->
```

Note

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" on page 56.

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.

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.

Note

The ASP Server caches the contents of ASP pages and the pages they include (known as include files). In previous versions of Sun ONE ASP (Sun Chili!Soft ASP), once an ASP page with include files had been cached, the only way to refresh it was to change the top ASP page or reset the ASP application. This functionality has changed. Now, when an include file is changed, the ASP Server automatically refreshes the cache the next time the file is accessed.

Note

Do not use an @LANGUAGE directive in a server-side include unless you are certain that an @LANGUAGE directive has not been used in the parent file. The existence of two @LANGUAGE directives could produce an error.

See also:

"Creating the Basic ASP Application" on page 182

"Creating an ASP Page" on page 183

"Using @Directives" on page 186

Defining the 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 ONE 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" on page 47.

See also:

"Using the Global.asa File" on page 189

"Creating the Basic ASP Application" on page 182

"Creating an ASP Page" on page 183

Using the Global.asa File

The global.asa file 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 top-level directory containing all application files and sub-directories).

The only script procedures you can declare in the global.asa file are listed below, and described in this section:

- **Application_OnStart** event
- **Application_OnEnd** event
- **Session_OnStart** event
- **Session_OnEnd** event

Note

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.

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 (on Windows), or when the ASP Server is shut down (on UNIX and Linux). 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 Sun ONE ASP VBScript or Sun ONE ASP JavaScript. 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:

"Chapter 15, Scripting Languages Reference" on page 503

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 Sun ONE ASP VBScript or Sun ONE ASP JavaScript. 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" on page 192

"Using the Global.asa File" on page 189

"Saving Changes to the Global.asa File" on page 194

"Chapter 15, Scripting Languages Reference" on page 503

Managing User Sessions

Communication between a browser and a Web server uses the HTTP protocol. This protocol is stateless, meaning that no information is 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 ONE Active Server Pages 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 session tracking off for a page. If your page does not rely on session information, turning session tracking off can decrease the time it takes Sun ONE ASP to process the script. By default, sessions are enabled.

The syntax is as follows:

```
<%@ ENABLESESSIONSTATE=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. With the **Session_OnStart** event, when session-wide variables are set, 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 Sun ONE ASP VBScript or Sun ONE ASP JavaScript. 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:

"Chapter 15, Scripting Languages Reference" on page 503

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 Sun ONE ASP VBScript or Sun ONE ASP JavaScript. 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:

"Using @Directives" on page 186

"Specifying Application Events" on page 190

"Using the Global.asa File" on page 189

"Saving Changes to the Global.asa File" on page 194

"Chapter 15, Scripting Languages Reference" on page 503

Saving Changes to the Global.asa File

When you make changes to the global.asa file and then save them, the Sun ONE ASP Server reloads and compiles the 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" on page 189

"Specifying Application Events" on page 190

"Managing User Sessions" on page 192

Using Sun ONE ASP Built-in Objects

Sun ONE Active Server Pages includes built-in objects that handle many common programming tasks. These objects enable you to avoid much of the overhead associated with complex Web programming. The following table lists and briefly describes the built-in objects included with Sun ONE ASP. Complete reference information can be found in "Chapter 9, ASP Built-in Objects Reference" on page 215.

Object	Description
"ASP Application Object" on page 216	The **Application** object shares application-level information and control settings for the lifetime of the 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.
"ASPError Object" on page 222	The **ASPError** object reports error information, and can be used to obtain information about an error condition that has occurred in script in an ASP page. The **ASPError** object is returned by the **Server.GetLastError** method. It has no methods but exposes several read-only properties, which provide specific information about the error the object represents.
"ASP Request Object" on page 224	The **Request** object is used to get information from the user that is passed along in an HTTP request.
"ASP Response Object" on page 235	The **Response** object is used to send information to the user.

Object	Description
"ASP Server Object" on page 251	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.
"ASP Session Object" on page 261	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.

Sun ONE ASP objects use methods to perform some type of procedure, and properties to store object attributes (such as color, font, or size). Some of the objects also contain collections (bits of information that are accessed in the same way).

For Web applications requiring more powerful programming, Sun ONE ASP uses Java as the primary method for extending the ASP architecture into the enterprise environment. Sun ONE ASP supports JavaBeans and Enterprise JavaBeans (EJB), as well as Common Object Request Broker Architecture (CORBA) objects. Component Object Model (COM) objects can also be used to process data and deliver output, with scripts acting as the "glue" to link COM objects.

Using Sun ONE ASP Installed Components

In addition to the built-in objects described in "Using Sun ONE ASP Built-in Objects" on page 194, Sun ONE Active Server Pages automatically installs a number of components that you can use to build dynamic Web pages. The following table lists and briefly describes these installed components. For more information about using them, see "ASP Component Reference" on page 271.

Component	Description
"ASP Ad Rotator Component" on page 272	Creates an **AdRotator** object that automates the rotation of advertisement images on a Web page.
"ASP Browser Capabilities Component" on page 278	Creates a **BrowserType** object that determines the type, version, and capabilities of every browser that visits your site.
"ASP Content Linking Component" on page 282	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.
"ASP Content Rotator Component" on page 288	Creates a **ContentRotator** object that automatically rotates HTML content strings on a Web page.

Component	Description
"ASP Counters Component" on page 293	Creates a **Counters** object that can create, store, increment, and retrieve any number of individual counters.
"ASP MyInfo Component" on page 296	Creates a **MyInfo** object that keeps track of personal information, such as the site administrator's name, address, and display choices.
"ASP Tools Component" on page 297	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 Sun ONE ASP Chili!Beans, Sun ONE Active Server Pages 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 COM controllers (such as ActiveX scripting engines like VBScript). Chili!Beans is designed to work with Java virtual machine (JVM) versions 1.4 or greater.

For more information, see "Chili!Beans Component Reference" on page 465.

Using Custom Server Components

On Windows and UNIX systems, Sun ONE Active Server Pages provides support for custom server components, which can be useful when your Web applications require complex business logic. Application developers may find it more efficient to encapsulate this business logic in a custom server component written in an advanced language such as Java or C++ (on Windows), rather than trying to implement it with script. Sun ONE ASP uses Java as the primary method for extending the ASP architecture into the enterprise environment.

Note

Sun ONE 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 ONE ASP Chili!Beans technology instead.

On Windows, Sun ONE ASP server components can be written in any language that supports COM, including Java, Visual Basic, and C++. On UNIX, Sun ONE ASP server components can be written in Java. If you develop your components in Java, Chili!Beans shields you from many of the details of COM. For more information, see "Chili!Beans Component Reference" on page 465.

Connecting to a Database

This section takes 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 ONE Active Server Pages, and how to migrate a Microsoft Access database running on a Windows-based computer to a MySQL or dBASE database running on a UNIX or Linux system (detailed information about migrating an Access database to MySQL is also provided in "Chapter 7, Using Database Tools" on page 135).

A connection string provides information required by the Sun ONE 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.

A database connection is opened by using the ADO **Connection** object included with Sun ONE 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" on page 301.

Note

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 Sun ONE ASP Configuration File" on page 517 (see the `[ADO]` keyword).

In this section:

Creating Connection Strings

When you want to connect to a database from an ASP page, the 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 ONE Active Server Pages supports the following three approaches to specifying parameters in a connection string:

- **System DSN:** 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.

- **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.
- **DSN-less connection strings:** With a DSN-less connection string, you specify all of the required database information in the connection string.

This section describes each approach listed above. In this section:

"Using System DSNs" on page 199

"Using DSN-less Connection Strings" on page 200

"Syntax for DSN-less Connection Strings" on page 201

"Using File DSNs" on page 203

"Parameters for File DSNs" on page 205

Note

Connection strings must be constructed according to the requirements of the ODBC driver being used. Sun ONE 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 ONE Active Server Pages for UNIX or Linux, you may need 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" on page 208.

Note

On UNIX and Linux systems, Sun ONE 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, see "Supported in This Release" on page 5.

Using System DSNs

As discussed in "Creating Connection Strings" on page 197, 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" on page 106. This saves information on the ASP Server about all parameters required for connecting to the database.

Note

On UNIX and Linux systems, Sun ONE Active Server Pages 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 "Supported in This Release" on page 5.

Once a system DSN is configured, rather than specifying 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" on page 203 and "Using DSN-less Connection Strings" on page 200. 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" on page 197

"Using FrontPage Database Features" on page 209

Using DSN-less Connection Strings

As discussed in "Creating Connection Strings" on page 197, using a DSN-less connection is one way to specify the information (in the form of parameters and their values) that is needed to establish 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];
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 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.

Different types of databases can require that you specify different parameters. The parameters to configure for each database in DSN-less connection strings are provided in "Syntax for DSN-less Connection Strings" on page 201.

Connection strings must be constructed according to the requirements of the ODBC driver being used. Sun ONE 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 ONE ASP for UNIX or Linux, you may need to edit them to use the syntax described in this section.

With Sun ONE 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.

Note

On UNIX and Linux systems, Sun ONE 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, see "Supported in This Release" on page 5.

See also:

"Connecting to a Database" on page 197

"Creating Connection Strings" on page 197

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.

Type	Syntax
DB2	`connect_string = "Driver={DB2}; IpAddress=[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.

Type	Syntax

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.

Type	Syntax

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" on page 200

Using File DSNs

As discussed in "Creating Connection Strings" on page 197, using file DSNs is one way to specify the information needed for establishing a connection to a database from an ASP application. This section 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 that must be configured for each database are provided in "Parameters for File DSNs" on page 205.

Note

File DSNs and connection strings must be constructed according to the requirements of the ODBC driver being used. On Windows, Sun ONE Active Server Pages 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 ONE 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.

When finished defining parameters, give the file a DSN file name 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

On UNIX and Linux systems, Sun ONE 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, see "Supported in This Release" on page 5.

You cannot use DSN-less connection strings or file DSNs for connecting to Microsoft Access or Microsoft SQL Server 6.5 databases from Sun ONE ASP for UNIX or Linux; you must use system DSNs.

See also:

"Connecting to a Database" on page 197

"Creating Connection Strings" on page 197

"Using System DSNs" on page 199

"Using FrontPage Database Features" on page 209

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.

Type	Parameters

DB2	`Driver={DB2}`
	`IpAddress=[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.

Type	Parameters

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.

Type	Parameters

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.

Type	Parameters

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_location]`
	where `[database_location]` is the directory in which the text files are stored.

See also:

"Using File DSNs" on page 203

Opening the Database Connection

Sun ONE Active Server Pages includes an 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.

"Creating Connection Strings" on page 197 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 ONE ASP to open a database connection.

Note

On UNIX and Linux systems, Sun ONE 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, see "Supported in This Release" on page 5.

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" on page 301.

If desired, you can also use FrontPage to create connection strings and display data on a Web page, as described in "Using FrontPage Database Features" on page 209.

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. Also, be sure to test the driver on a nonproduction server. A malfunctioning ODBC driver can bring down your ASP Server.

See also:

"Connecting to a Database" on page 197

Using FrontPage Database Features

This section describes Sun ONE Active Server Pages support for the database connectivity features of FrontPage.

Note

While Sun ONE ASP enables you to run ASP pages generated by FrontPage, specific questions about the installation, configuration, and use of FrontPage and FrontPage Server Extensions should be directed to Microsoft or its representatives.

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 ONE Active Server Pages for UNIX or Linux, you may need to first edit them so they use the syntax described in "Creating Connection Strings" on page 197.

In addition, when you use Microsoft SQL Server 6.5 or Microsoft Access databases with Sun ONE 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" on page 128.

When you are using Sun ONE ASP for UNIX or Linux, consider migrating your Microsoft Access databases to MySQL, as described in "Database Publisher" on page 135, or to dBASE, as described in "Migrating an Access Database to MySQL or dBASE" on page 211. 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" on page 197.

For all other databases supported by Sun ONE ASP, you can use system DSNs, file DSNs, and DSN-less connection strings.

See also:

"Using FrontPage Database Features" on page 209

"Connecting to a Database" on page 197

Displaying Data on a Web Page with FrontPage

With Sun ONE Active Server Pages, FrontPage developers can continue to use FrontPage features for connecting to a database and displaying its information on an ASP page. Sun ONE 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 ONE 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.

Note

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 ONE ASP for UNIX or Linux. For more information, see "Using FrontPage Database Connections" on page 209.

See also:

"Using FrontPage Database Connections" on page 209

"Connecting to a Database" on page 197

Migrating an Access Database to MySQL or dBASE

Microsoft Access databases are compatible with Sun ONE Active Server Pages running on Windows, but these databases do not run on UNIX or Linux systems. There are several options for connecting to a Microsoft Access database with Sun ONE ASP for UNIX or Linux, as described below.

- Use SequeLink. The necessary steps for creating the connection string are described in "Creating Connection Strings" on page 197. The steps the system administrator must take are described in "Configuring SequeLink" on page 128.
- Use the Sun ONE ASP Database Publisher tool to migrate an Access database to MySQL. For more information, see "Database Publisher" on page 135.
- 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 ONE 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" on page 197

"Using FrontPage Database Features" on page 209

Developing International Applications

The default locale is selected during the installation of Sun ONE Active Server Pages. If you want to deliver ASP applications in different locales and languages, your administrator can change the locale specified for the ASP Server, as described in "Configuring International Support" on page 43. 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
%>
```

You can also change code pages by using the **Session.CodePage** property. For more information about the **Session.LCID** and **Session.CodePage** properties, see "ASP Session Object Properties" on page 264.

Japanese Character Support

Sun ONE Active Server Pages 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 an ASP page, including literal strings in the source files, text stored to files via the **Scripting.FileSystemObject**, and text stored to databases via the various ADODB objects and methods. (The implementation of ADO used with Sun ONE 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.

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 ONE 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" on page 213

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 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" on page 212

Publishing a Sun ONE 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 ASP Applications" on page 48. For more information about how ASP applications are structured, see "Creating the Basic ASP Application" on page 182.

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.

Note

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 about the **EnableParentPaths** setting, see "Configuring File System Access" on page 56.

9 ASP Built-in Objects Reference

Sun ONE Active Server Pages includes built-in or intrinsic objects that handle many common programming tasks. The objects included in Sun ONE ASP are **Application**, **ASPError**, **Request**, **Response**, **Server**, and **Session**. Built-in objects are included on all ASP pages, and do not need to be created before they can be used. These objects enable you to avoid much of the overhead associated with complex Web programming, simplifying development by solving Web-protocol programming issues.

The built-in objects and their roles are listed in the following table, and discussed in this chapter. This chapter provides basic reference information about the ASP intrinsic objects. You may wish to consult additional print and Web resources for more detailed ASP reference information.

Note

ASP intrinsic objects can now be accessed directly from Java code using Sun ONE ASP Chili!Beans technology. For more information, see "ASP Servlet Interface" on page 472.

Object	Description
"ASP Application Object" on page 216	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.
"ASPError Object" on page 222	Reports error information.
"ASP Request Object" on page 224	Provides access to values passed to the server by the client.
"ASP Response Object" on page 235	Controls the output from an ASP script to the requesting client.
"ASP Server Object" on page 251	Provides access to methods and properties on the server. These methods and properties typically serve as utility functions.
"ASP Session Object" on page 261	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, but instead persists for the entire user session.

Note

ASP scripts provided in examples are assumed to be enclosed in script delimiters. The `<% %>` and `<SCRIPT>` delimiters are generally not shown.

See also:

"Using Sun ONE ASP Built-in Objects" on page 194

ASP Application Object

The ASP **Application** object shares information among all users of a given application. An ASP-based application is defined as all .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 cannot alter a property simultaneously.

Syntax: ASP Application Object

`Application.`*method*

ASP Application Object Collections

ASP **Application** object collections are listed in the following table.

Collection	Description
"ASP Application Object Contents Collection" on page 216	Contains all items added to the application through script commands.
"ASP Application Object StaticObjects Collection" on page 217	Contains all objects added to the session with the `<OBJECT>` tag.

ASP Application Object Contents Collection

The **Application.Contents** collection contains all items added to the application through script commands. The **Contents** collection can be used to obtain a list of items that have been given application scope, or to specify an item to be the target of an operation. The **Contents.Remove** and **Contents.RemoveAll** methods can be used to remove some or all of the items from the collection.

Syntax: ASP Application Object Contents Collection

`Application.Content``(`*key*`)`

Parameters: ASP Application Object Contents Collection

key

The name of the item to retrieve.

Remarks: ASP Application Object Contents Collection

The **Application.Contents** collection contains all items that have been declared at the application level without using the `<OBJECT>` tag. This includes objects created with **Server.CreateObject**, and scalar variables established through an **Application** declaration.

Example: ASP Application Object Contents Collection

In the following script, `objFSO`, `strHello`, and `Start_Time` are members of the **Application.Contents** collection:

```
<%
Set Application("objFso") =
Server.CreateObject("Scripting.FileSystemObject")
Application("strHello") = "Hello"
Application("Start_Time") = CStr(Now)
%>
```

The **Application.Contents** collection supports `For...Each` and `For...Next` iteration, as illustrated below.

```
<%
  For Each Key in Application.Contents
    Response.Write Key + " = " + Application(Key) + "<BR>"
  Next
%>


<%
  For intItem = 1 to Application.Contents.Count
    Response.Write CStr(intItem) + " = "
    Response.Write Application.Contents(intItem) + "<BR>"
  Next
%>
```

ASP Application Object StaticObjects Collection

The **Application.StaticObjects** collection contains all objects added to the session with the `<OBJECT>` tag. The collection can be used to retrieve the value of a specific property for an object, or to retrieve all properties for all static objects.

Syntax: ASP Application Object StaticObjects Collection

Application.StaticObjects(*key*)

Parameters: ASP Application Object StaticObjects Collection

key

The name of the item to retrieve.

Remarks: ASP Application Object StaticObjects Collection

An iterating control structure can be used to loop through the keys of the **StaticObjects** collection.

Example: ASP Application Object StaticObjects Collection

An interating control structure can be used to loop through the keys of the **StaticObjects** collection, as shown in the following example.

```
<%
  Dim Key


  For Each Key In Application.StaticObjects
    Response.Write Key & " = <i>(object)</i><BR>"
  Next
%>
```

ASP Application Object Methods

ASP **Application** object methods are listed in the following table.

Method	Description
"ASP Application Contents.Remove Method" on page 219	Removes a single item from the **Application** object **Contents** collection.
"ASP Application Contents.RemoveAll Method" on page 220	Removes all items from the **Application** object **Contents** collection.
"ASP Application Object Lock Method" on page 220	Prevents script from modifying object properties.
"ASP Application Object Unlock Method" on page 220	Enables script to modify object properties after execution of the **Lock** method.

ASP Application Contents.Remove Method

The **Application.Contents.Remove** method removes a single application variable from the **Application** object **Contents** collection.

Syntax: ASP Application Object Contents.Remove Method

`Application.Contents.Remove`(*name* | *index*)

Parameters: ASP Application Object Contents.Remove Method

name

The identifier for the application variable to remove.

index

An index offset indicating which application variable in the list to remove.

Remarks: ASP Application Object Contents.Remove Method

The **Contents.Remove** method takes a string or an integer as an input parameter. If the input parameter is a string, the method searches the **Contents** collection for an application variable with that name and removes it. If the input parameter is an integer, the method counts that number of application variables from the start of the collection, and removes the corresponding variable.

Example: ASP Application Object Contents.Remove Method

The following code includes members of the **Application.Contents** collection:

```
<%
Set Application("objFso") =
Server.CreateObject("Scripting.FileSystemObject")
Application("strHello") = "Hello"
Application("Start_Time") = CStr(Now)
%>
```

The following code removes the second item in the collection (`"strHello"`) using an integer.

```
<%
Application.Contents.Remove(2)
%>
```

The following code removes the second item in the collection (`"strHello"`) using the variable name.

```
<%
Application.Contents.Remove("strHello")
%>
```

ASP Application Contents.RemoveAll Method

The **Application.Contents.RemoveAll** method removes all application variables from the **Application** object **Contents** collection.

Syntax: ASP Application Object Contents.RemoveAll Method

```
Application.Contents.RemoveAll()
```

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 the **Unlock** method is not called 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 the **Unlock** method is not called explicitly, the server unlocks the locked **Application** object when the script ends or times out.

Syntax: ASP Application Object Unlock Method

```
Application.Unlock
```

ASP Application Object Events

ASP **Application** object events are listed in the following table.

Event	Description
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 on Windows, and when the ASP Server is shut down on UNIX and Linux.

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

If you store an array in the **Application** object, it is strongly recommended that you retrieve a copy of the array before retrieving or changing any 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 that 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
```

ASPError Object

The **ASPError** object reports error information, and can be used to obtain information about an error condition that has occurred in script in an ASP page. The **ASPError** object is returned by the **Server.GetLastError** method. It has no methods but exposes several read-only properties, which provide specific information about the error the object represents.

Syntax: ASPError Object

`ASPError.`*property*

ASPError Object Properties

The **ASPError** object has no methods but exposes several read-only properties, which provide specific information about the error the object represents. Those properties are listed in the following table.

Property	Description
ASPCode	Returns an error code generated by IIS.
Number	Returns the standard COM error code.

Property	Description
Source	Returns the actual source code (when available) of the line that caused the error.
Category	Indicates if the source of the error was internal to ASP, to the scripting language, or to an object.
File	Indicates the name of the .asp file that was being processed when the error occurred.
Line	Indicates the line within the .asp file that generated the error.
Column	Indicates the column position within the .asp file that generated the error.
Description	Returns a short description of the error.
ASPDescription	Returns a more detailed description of the error if it relates to ASP.

ASPError Object Example

The following example illustrates information exposed by the **ASPServer** object:

```
<%
'The following line of code will give an error
Set Application("objFso") = Server.CreateObject("")

'Call the GetLastError() method to trap the error
set objErr=Server.GetLastError()

Response.Write("ASP Code=" & objErr.ASPCode & "<br>")
Response.Write("Number=" & objErr.Number & "<br>")
Response.Write("Source=" & objErr.Source & "<br>")
Response.Write("Category=" & objErr.Category & "<br>")
Response.Write("File=" & objErr.File & "<br>")
Response.Write("Line=" & objErr.Line & "<br>")
Response.Write("Column=" & objErr.Column & "<br>")
Response.Write("Description=" & objErr.Description & "<br>")
Response.Write("ASPDescription=" & objErr.ASPDescription & "<br>")
%>
```

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 collections are listed in the following table.

Collection	Description
"ASP Request Object Cookies Collection" on page 224	The value of cookies sent in the HTTP request.
"ASP Request Object Form Collection" on page 226	The values of form elements sent in the HTTP request body.
"ASP Request Object QueryString Collection" on page 228	The value of variables in the HTTP query string.
"ASP Request Object ServerVariables Collection" on page 229	The value of predetermined environment variables.

Note

Due to widely differing Web-server support for client-side certificates, Sun ONE ASP does not implement the **ClientCertificate** collection.

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 as follows: **HasKeys**, which is read-only and 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 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`.

An iterator can be used to cycle through all cookies in the **Cookie** collection, or all 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``(`*element*`) [(`*index*`) | .Count]`

Parameters: ASP Request Object Form Collection

element

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 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 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 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

An iterator can be used to loop through all 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 listed in 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 ONE ASP server.
ASP_VERSION_MAJOR	The major version number of the Sun ONE ASP server.
ASP_VERSION_MINOR	The minor version number of the Sun ONE ASP server.
ASP_OS	The operating system the server is running on.
ASP_LICENSE	License information for the Sun ONE 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*.

Value	Description
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. **Note:** Not supported by Sun ONE ASP for UNIX.
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 ONE 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.

Value	Description
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 ONE 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

An iterator can be used to loop through each server variable name. For example, the following script prints all 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 Properties

The ASP **Request** object has one property, which is listed below.

"ASP Request Object TotalBytes Property" on page 233	The total number of bytes the client is sending in the body of the request.

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 Request Object Methods

The ASP **Request** object has one method, which is listed below.

"ASP Request Object BinaryRead Method" on page 234	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 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 Response Object

The **Response** object controls sending output to the browser.

Syntax: ASP Response Object

Response.*collection* | *property* | *method*

ASP Response Object Collections

The ASP **Response** object has one collection, which is listed below.

"ASP Response Object Cookies Collection" on page 235	You can use this collection to set cookie values to send to the client browser.

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. This can be one of the attributes listed in the following table.

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`.

An iterator can be used to set cookie attributes. The following example sets all cookies in a collection to expire on Dec. 31, 2003:

```
<%
For each cookie in Response.Cookies
   Response.Cookies(cookie).ExpiresAbsolute = #Dec. 31, 2002#
Next
%>
```

An iterator can also be used to set the values of all cookies in a collection, or all 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 Properties

ASP **Response** object properties are listed in the following table.

Property	Description
"ASP Response Object Buffer Property" on page 238	Indicates whether to buffer page output.
"ASP Response Object CacheControl Property" on page 239	Determines if proxy servers are allowed to cache the output generated by ASP.
"ASP Response Object Charset Property" on page 239	Appends the name of the character set to the Content-Type header.
"ASP Response Object CodePage Property" on page 240	Sets the code page for one response.
"ASP Response Object ContentType Property" on page 241	Specifies the HTTP content type for the response.
"ASP Response Object Expires Property" on page 241	Specifies the length of time until the page cached on a browser expires.
"ASP Response Object ExpiresAbsolute Property" on page 242	Specifies the date and time a page cached on a browser expires.
"ASP Response Object IsClientConnected Property" on page 243	Indicates if the client is still connected to the server.
"ASP Response Object LCID Property" on page 243	Sets the LCID for data for one response.
"ASP Response Object PICS Property" on page 244	Adds the value of a PICS (Platform for Internet Content System) label to the pics-label field of the response header.
"ASP Response Object Status Property" on page 245	The value of the status line returned by the server.

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 **Response** object **Flush** or **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 values listed below:

TRUE	Output is buffered. The server does not send output from the script on the page until all of the script has been processed 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 **Flush** method, the server will honor Keep-Alive requests made by the client. This saves time because the server does not need 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 CodePage Property

Response.CodePage sets the code page for one response. Code pages tell the server how to encode characters for different languages. The code page is a numeric value of the character set. Different languages and locales may use different code pages.

Syntax: ASP Response Object CodePage Property

Response.CodePage [= *CodePageID*]

Parameters: ASP Response Object CodePage Property

CodePageID

An integer that represents the character formatting code page. For a list of the code page integers for the languages supported by Sun ONE ASP, see "Configuring International Support" on page 43.

Remarks: ASP Response Object CodePage Property

Response.CodePage explicitly affects a single page, whereas **Session.CodePage** affects all responses in a session.

If the code page is set in a page, then **Response.Charset** should also be set. The code page value tells the server how data should be encoded when building the response, and the charset value tells the browser how the data should be decoded when displaying the response. *CharsetName* of **Response.Charset** must match the code page value. If they do not match, mixed characters will be displayed in the browser.

"ASP Session Object CodePage Property" on page 264

"ASP Response Object Charset Property" on page 239

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 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 zero (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

```
Response.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, 2003:

```
Response.ExpiresAbsolute = #May 31, 2003 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**.

Note

Response.IsClientConnected is not supported in this release of Sun ONE ASP.

Syntax: ASP Response Object IsClientConnected Property

```
Response.IsClientConnected()
```

Remarks: ASP Response Object IsClientConnected Property

This property allows you greater control over circumstances in which 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 LCID Property

The **Response.LCID** property enables you to set or get a Locale Identifier (LCID) that specifies how dates, times, and currencies are formatted for specific geographic locales. **Response.LCID** sets the LCID for data for one response.

Syntax: ASP Response Object LCID Property

Response.LCID [= *LCIDnumber*]

Parameters: ASP Response Object LCID Property

LCIDnumber

A valid Locale Identifier (LCID) number. For a list of valid values, see "Configuring International Support" on page 43.

Remarks: ASP Response Object LCID Property

Response.LCID sets the LCID for one response, whereas **Session.LCID** sets the LCID for an entire session. For usage and limitations, see "Developing International Applications" on page 212.

See also:

"ASP Session Object LCID Property" on page 265

ASP Response Object PICS Property

The **PICS** property adds a value to the pics-label field of the response header (PICS stands for Platform for Internet Content System).

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, quotes must be replaced 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 Response Object Methods

ASP **Response** object methods are listed in the following table.

Method	Description
"ASP Response Object AddHeader Method" on page 246	Set the HTML header *name* to *value*.
"ASP Response Object AppendToLog Method" on page 248	Adds a string to the end of the Web server log entry for this request.
"ASP Response Object BinaryWrite Method" on page 248	Writes the given information to the current HTTP output without any character set conversion.
"ASP Response Object Clear Method" on page 249	Erases any buffered HTML output.
"ASP Response Object End Method" on page 249	Stops processing the .asp file and returns the current results.
"ASP Response Object Flush Method" on page 249	Sends any buffered HTML output immediately.
"ASP Response Object Redirect Method" on page 250	Sends a redirect message to the browser, causing it to attempt to connect to a different URL.
"ASP Response Object Write Method" on page 250	Writes a variable to the current HTML output as a string.

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 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 **Response** object **Write** method) is sent to the client. The exception to this rule is when the **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 **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 was 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 the **BinaryWrite** method:

```
<%
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 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

```
Response.Redirect URL
```

Parameters: ASP Response Object Redirect Method

URL

The Uniform Resource Locator the client is redirected to.

Remarks: ASP Response Object Redirect Method

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 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.property` *property* | *method*

ASP Server Object Properties

The ASP **Server** object has one property, which is listed below.

"ASP Server Object ScriptTimeout Property" on page 252	The length of time a script runs before it is terminated.

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 90 seconds by default. 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* was 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 Server Object Methods

ASP **Server** object methods are listed in the following table.

Method	Description
"ASP Server Object CreateObject Method" on page 253	Creates an instance of a server component.
"ASP Server Object Execute Method" on page 254	Executes an .asp file.

Method	Description
"ASP Server Object GetLastError Method" on page 255	Returns an **ASPError** object that describes the error condition.
"ASP Server Object HTMLEncode Method" on page 258	Applies HTML encoding to a specified string.
"ASP Server Object MapPath Method" on page 258	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 Transfer Method" on page 260	Sends all current state information to another .asp file for processing.
"ASP Server Object URLEncode Method" on page 261	Applies URL encoding rules, including escape characters, to a string.

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 Execute Method

The **Execute** method calls an .asp file and processes it as if it were part of the calling ASP script (similar to a procedure call in many programming languages).

Syntax: ASP Server Object Execute Method

Server.Execute(*path*)

Parameters: ASP Server Object Execute Method

Server.Execute takes a single parameter, which is the path to another .asp page.

path

A string that specifies the location of the .asp file to execute. The parameter is either an absolute or relative path. If the path is absolute, *path* must map to an ASP script in the same application as the calling .asp file. The *path* parameter can be a string variable name that is set at run-time.

Remarks: ASP Server Object Execute Method

The **Server.Execute** method provides a way to divide complex applications into individual modules. You can use **Server.Execute** to run ASP code on another page, returning to the original page when the code has finished executing.

By employing the **Server.Execute** method, a library of .asp files can be developed and then called as needed. This approach is similar to server-side includes, with the only major difference being that **Server.Execute** allows you to dynamically call an .asp file. Because **Server.Execute** is an ASP method rather than an HTML comment, it can be used to conditionally execute scripts and avoid including huge include files. Server-side includes are executed prior to any ASP code and thus can't be executed conditionally.

Server.Execute does have some potential drawbacks, which are listed below. In many situations, though, using **Server.Execute** is a better choice than using server-side includes.

Potential drawbacks to **Server.Execute**:

- **Server.Execute** cannot call a particular procedure on the included page. It starts running code at the top of the executed page and continues until the executed page ends, at which point it returns to the original page.
- Page-scope variables are not shared between the original page and the executed page. This isn't an issue with server-side includes because the include file code is incorporated into the original page before being executed.

Examples: ASP Server Object Execute Method

The following code illustrates use of the **Server.Execute** method:

File1.asp:

```
<%
Response.Write("This is File 1!<br>")
Server.Execute("file2.asp")
Response.Write("This is File 1 again!")
%>
```

File2.asp:

```
<%
Response.Write("This is File 2!<br>")
%>
```

Output would be as follows:

This is File 1!

This is File 2!

This is File 1 again!

ASP Server Object GetLastError Method

The **GetLastError** method returns an **ASPError** object that describes the error condition, and is available only before the .asp file has sent content to the client. The **ASPError** object has a number of properties that return information about the most recent error that occurred before **GetLastError** was called.

See also:

"ASPError Object" on page 222

Syntax: ASP Server Object GetLastError Method

```
Server.GetLastError()
```

Remarks: ASP Server Object GetLastError Method

The correct functionality of the **Server.GetLastError** method depends on the behavior of the Web server. Because the dependent behavior is implemented only on Microsoft's IIS Web server, and not on other Web servers such as the Apache Web Server or Sun ONE Web Server (formerly iPlanet), Sun ONE ASP is not in complete compliance with Microsoft ASP 3.0 functionality for this method.

On IIS, when an ASP page gets an unhandled error (parsing or run time), page control is transferred to a predefined error page. For 500:100 errors, this page is another ASP page that displays specific information about the error that was received. The error-handling page uses the **Server.GetLastError** method to retrieve the **ASPError** object to access specific information about the error that just occurred. When the unhandled error occurs, IIS relies on the **Server.Transfer** mechanism built into the ASP server to transfer control to the error-handling page.

Because the Sun ONE and Apache Web servers use their own mechanisms for displaying unhandled errors, that functionality is implemented inside the ASP Server. To implement this functionality, a directive has been added to the Sun ONE ASP Server's configuration file (casp.cnfg). The directive tells the ASP server where to transfer control when an unhandled error occurs. The name of the new directive is `500errordocument=<`*path*`>` and it is located in the `[default application]` section in casp.cnfg. There are two possible values for this directive. If the directive contains the path to a valid ASP or HTML file, that file will be returned when the ASP server encounters a 500 error. If the directive is left blank, the ASP server uses a custom 500 error page installed with the product.

The custom page installed with Sun ONE ASP is identical to the custom page installed with Microsoft IIS 5.0. When an unhandled error occurs on the Sun ONE ASP server, control is transferred (using the **Server.Transfer** method) to the page listed in casp.cnfg. This page uses the **Server.GetLastError** method to display detailed information about the error that just occurred.

In addition, a `500-15errordocument=<`*path*`>` directive has also been added to casp.cnfg. When the browser requests the global.asa file and an error is encountered, the server redirects to this custom error page.

For more information about these casp.cnfg settings, see the settings listed in "[default application]" on page 521.

Examples: ASP Server Object GetLastError Method

The following examples demonstrate different sorts of errors that will generate a 500:100 custom error. The three types of errors are:

- Pre-processing errors
- Syntax errors
- Run-time errors

Example 1

The following example produces a pre-processing error. The error will be generated because the `#include` statement is missing the file parameter for the statement.

```
<!--#include f="header.inc" -->
<%
Response.Write("sometext")
%>
```

Example 2

The following example produces VBScript syntax error 1028: `"Expected 'While', 'Until' or end of statement."`

```
<%
    Dim A

    A = 0
    i = 0
    Do i < 10
       A = A + i
       i = i + 1
    Loop
%>
```

Example 3

The following example produces VBScript runtime error 450: `"Wrong number of arguments or invalid property assignment."`

```
<%
Dim A

A = 10

Function foo( x )
    foo = x + 1
End Function

A = foo()
%>
```

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, 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 Transfer Method

The **Transfer** method sends all current state information to another .asp file for processing.

Syntax: ASP Server Object Transfer Method

Server.Transfer(*path*)

Parameters: ASP Server Object Transfer Method

path

The location of the .asp file to which control should be transferred.

Remarks: ASP Server Object Transfer Method

The **Transfer** method is an alternative to using **Response.Redirect**. When **Server.Transfer** is called, state information for all ASP built-in objects is included in the transfer from one .asp file to another. The transfer takes place on the server, instead of forcing the browser to redirect to a new page.

When **Server.Transfer** is called, execution of the first page is terminated and execution of the second page begins. If the first page has started writing to the response buffer, the second page appends to the buffer instead of replacing it. If buffering is on, then HTTP headers can be modified by the ASP file receiving the transfer. If buffering is off, the HTTP headers are not modifiable by the ASP file receiving the transfer, unless content has not yet been sent by ASP. Multiple transfers can be called in succession, thus chaining pages together.

The only data transferred to a second ASP page is the ASP built-in objects and **ASPError** object values from the first request. Variables declared by the first ASP page are not available in the second ASP page.

When you transfer to a page in another application, the **Application** and **Session** objects contain information from the originating application. Accordingly, the ASP page receiving the transfer is treated as part of the originating application.

Examples: ASP Server Object Transfer Method

The following code illustrates use of the **Server.Transfer** method:

File1.asp:

```
<%
Response.Write("This is line 1 in File1.asp<br>")
Server.Transfer("file2.asp")
```

```
Response.Write("This is line 2 in File1.asp")
%>
```

File2.asp:

```
<%
Response.Write("This is line 1 in File2.asp<br>")
Response.Write("This is line 2 in File2.asp<br>")
%>
```

Output would be as follows:

This is line 1 in File1.asp

This is line 1 in File2.asp

This is line 2 in File2.asp

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 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, but rather 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 collections are listed in the following table.

Collection	Description
"ASP Session Object Contents Collection" on page 262	Contains the items you have added to the session with script commands.
"ASP Session Object StaticObjects Collection" on page 263	Contains items created in global.asa using the <OBJECT> tag and given session scope.

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. The **Contents.Remove** and **Contents.RemoveAll** methods can be used to remove items from the collection.

Syntax: ASP Session Object Contents Collection

Session.Contents(*key*)

Parameters: ASP Session Object Contents Collection

key

The name of the item to retrieve.

Methods: ASP Session Object Contents Collection

ASP **Session** object **Contents** collection methods are listed in the following table.

Method	Description
"ASP Session Object Contents.Remove Method" on page 268	Removes an item from the collection.
"ASP Session Object Contents.RemoveAll Method" on page 269	Removes all items from the collection.

Remarks: ASP Session Object Contents Collection

An iterating control structure can be used 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 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 an iterator can be used to retrieve all 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

An iterating control structure can be used 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 Properties

ASP **Session** object properties are listed in the following table.

Property	Description
"ASP Session Object CodePage Property" on page 264	Sets the code page for an entire session.
"ASP Session Object LCID Property" on page 265	Sets or gets a Locale Identifier (LCID) that determines how certain content such as date, time, and currency is formatted.
"ASP Session Object SessionID Property" on page 265	Returns the session identifier for this client. Each session has a unique identifier.
"ASP Session Object Timeout Property" on page 266	The timeout period, in minutes, for session state for this application.

ASP Session Object CodePage Property

Session.CodePage sets the code page for an entire session. Code pages tell the server how to encode characters for different languages. The code page is a numeric value of the character set. Different languages and locales may use different code pages.

Syntax: ASP Session Object CodePage Property

`Session.CodePage` [= *CodePageID*]

Parameters: ASP Session Object CodePage Property

CodePageID

An integer that represents the character formatting code page. For a list of the code page integers for the languages supported by Sun ONE ASP, see "Configuring International Support" on page 43.

Remarks: ASP Session Object CodePage Property

Session.CodePage affects all responses in a session, whereas **Response.CodePage** affects a single response.

If the code page is set in a page, then **Response.Charset** should also be set. The code page value tells the server how data should be encoded when building the response, and the charset value tells the browser how the data should be decoded when displaying the response. *CharsetName* of **Response.Charset** must match the code page value. If they do not match, mixed characters will be displayed in the browser.

See also:

"ASP Response Object CodePage Property" on page 240

"ASP Response Object Charset Property" on page 239

ASP Session Object LCID Property

The **Session.LCID** property enables you to set or get a Locale Identifier (LCID) that specifies how dates, times, and currencies are formatted for specific geographic locales. **Session.LCID** sets the LCID for data for an entire session.

Syntax: ASP Session Object LCID Property

```
Session.LCID [= LCIDnumber]
```

Parameters: ASP Session Object LCID Property

LCIDnumber

A valid Locale Identifier (LCID) number. For a list of valid values, see "Configuring International Support" on page 43.

Remarks: ASP Session Object LCID Property

Session.LCID sets the LCID for an entire session, whereas **Response.LCID** sets the LCID for one response. For usage and limitations, see "Developing International Applications" on page 212.

See also:

"ASP Response Object LCID Property" on page 243

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.

Also, keep in mind that the session identifier persists as long as the current Web server session is running. If the Web server service is shut down, 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.

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 Methods

ASP **Session** object methods are listed in the following table.

"ASP Session Object Abandon Method" on page 267	Destroys a **Session** object and all objects stored in it, and releases their resources.
"ASP Session Object Contents.Remove Method" on page 268	Removes a single session variable from the **Session** object **Contents** collection.
"ASP Session Object Contents.RemoveAll Method" on page 269	Removes all session variables from the **Session** object **Contents** collection.

ASP Session Object Abandon Method

The **Abandon** method destroys all objects stored in a **Session** object and releases their resources. If the **Abandon** method is not called 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 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 Contents.Remove Method

The **Session.Contents.Remove** method removes a single session variable from the **Session** object **Contents** collection.

Syntax: ASP Session Object Contents.Remove Method

`Session.Contents.Remove`(*name* | *index*)

Parameters: ASP Session Object Contents.Remove Method

name

The identifier for the session variable to remove.

index

An index offset indicating which session variable in the list to remove.

Remarks: ASP Session Object Contents.Remove Method

The **Contents.Remove** method takes a string or an integer as an input parameter. If the input parameter is a string, the method searches the **Contents** collection for an application variable with that name and removes it. If the input parameter is an integer, the method counts that number of application variables from the start of the collection and removes the corresponding variable.

Example: ASP Session Object Contents.Remove Method

The following code includes members of the **Session.Contents** collection:

```
<%
Set Session("ASPAdRotator") = Server.CreateObject("MSWC.AdRotator")
Session("strHello") = "Hello"
Session("Start_Time") = CStr(Now)
%>
```

The following code removes the second item in the collection (`"strHello"`) using an integer.

```
<%
Session.Contents.Remove(2)
%>
```

The following code removes the second item in the collection (`"strHello"`) using the variable name.

```
<%
Session.Contents.Remove("strHello")
%>
```

ASP Session Object Contents.RemoveAll Method

The **Session.Contents.RemoveAll** method removes all session variables from the **Session** object **Contents** collection.

Syntax: ASP Session Object Contents.RemoveAll Method

```
Session.Contents.RemoveAll()
```

ASP Session Object Events

ASP **Session** object events are listed in the following table. **Session** object events are defined in the global.asa file.

Event	Description
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.

Remarks: ASP Session Object

Values can be stored 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
```

A built-in object cannot be stored 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 the 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 "ASP Application Object Examples" on page 221 for an example of storing an array.

10 ASP Component Reference

Sun ONE Active Server Pages automatically installs a number of components that can be used to build dynamic Web pages. The components installed with Sun ONE ASP are listed in the following table, and discussed in this chapter.

Note

This chapter provides basic reference information about ASP components. You may wish to consult additional print and Web resources for more detailed ASP reference information.

Component	Description
"ASP Ad Rotator Component" on page 272	Creates an **AdRotator** object that automates the rotation of advertisement images on a Web page.
"ASP Browser Capabilities Component" on page 278	Creates a **BrowserType** object that determines the type, version, and capabilities of every browser that visits your site.
"ASP Content Linking Component" on page 282	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.
"ASP Content Rotator Component" on page 288	Creates a **ContentRotator** object that automatically rotates HTML content strings on a Web page.
"ASP Counters Component" on page 293	Creates a **Counters** object that can create, store, increment, and retrieve any number of individual counters.
"ASP MyInfo Component" on page 296	Creates a **MyInfo** object that keeps track of personal information such as the site administrator's name, address, and display choices.
"ASP Tools Component" on page 297	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 **AdRotator** object that automates the rotation of advertisement images on a Web page. The **AdRotator** object relies on two additional files for parameters and functionaliy:

- The 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 (see "ASP Ad Rotator Component Rotator Schedule File" on page 272).
- The redirection file, an optional file that implements redirection, and enables you to record how many users click each advertisement (see "ASP Ad Rotator Component Redirection File" on page 275).

Each time a user opens or reloads the Web page, the **AdRotator** object displays a new advertisement based on the information specified in the rotator schedule file. The number of users who click each advertisement can be recorded by setting the URL parameter in the rotator schedule file to direct users to the redirection file. When this parameter is specified, each jump to an advertiser's URL is recorded in the Web server activity logs.

Registry Settings: ASP Ad Rotator Component

The Ad Rotator component does not use registry settings.

Syntax: ASP Ad Rotator Component

The Ad Rotator control is registered with the ProgId of "`MSWC.AdRotator`." The following VBScript excerpt creates an instance of the control.

```
Set adRot = Server.CreateObject("MSWC.AdRotator")
```

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 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 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, default values are used. 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

altText

impressions

Parameters: ASP Ad Rotator Component Rotator Schedule File

URL

Specifies the path to the dynamic-link library (.dll) or application file (.asp) that implements redirection.

This path can be specified 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.

altText

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.

Example: ASP Ad Rotator Component Rotator Schedule File

The following script demonstrates the use of a rotator schedule file to display a variety of advertisements, and how to include a redirection file.

```
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 he or she clicked.

Script can also be included in the redirection file to count the number of users that have clicked on a particular advertisement, and then save this information to a file on the server.

Example: 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

The Ad Rotator component exposes the properties listed below:

- **Border**
- **Clickable**
- **TargetFrame**

ASP Ad Rotator Component Border Property

The **Border** property specifies whether advertisements should be displayed with a 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 the rotator schedule file. 0 specifies no border.

ASP Ad Rotator Component Clickable Property

The **Clickable** property specifies whether advertisements are to be 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 be set to 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

The Ad Rotator component provides the following method:

- **GetAdvertisement**

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 is 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.

Example: 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") %>
```

Example: 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 output is:

```
<A HREF=
"/foo/bar.asp?url=http://www.chilisoft.com/info/index.html&image=/ads/
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")
```

The parameters are as follows:

BrowserType

Specifies the name of the object created by the call to **Server.CreateObject**.

Remarks: ASP Browser Capabilities Component

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	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

Property	Description
Version	Full version number of the browser

Browsecap.ini File: ASP Browser Capabilities Component

The browscap.ini file (called libchilicap.ini on UNIX) contains information about each known browser. It is a standard text file that lists the features a browser supports. The browscap.ini file maps browser capabilities to the HTTP User Agent header.

Be sure 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 it. 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 as "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 syntax for the default section of the browscap.ini file is as follows:

```
[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

This example shows some of the entries for Internet Explorer (IE) 5.0. Since it has no parent line, the only properties it has (other than those defined in the default section) are those explicitly defined:

```
[IE 5.0]
browser=IE
Version=5.0
majorver=#5
minorver=#0
frames=TRUE
tables=TRUE
cookies=TRUE
vbscript=TRUE
javascript=TRUE
ActiveXControls=TRUE
```

In the following example, IE 5.0 is specified as the parent for the browser. The properties explicitly provided will replace, or add to, those values in the parent's definition:

```
[Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)*]
parent=IE 5.0
platform=Windows 2000
version=5.0
minorver=01
[IE 5.0]
browser=IE
frames=TRUE
tables=TRUE
cookies=TRUE
backgroundsounds=TRUE
vbscript= TRUE
javascript= TRUE
. . .
```

To determine if a browser supports JavaScript, use the following code:

```
Set 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:

```
Set 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>JavaScript</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 pages in a Web site can be treated like pages in a book. The Content Linking component can be used 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 Content Linking component does not use registry settings.

Syntax: ASP Content Linking Component

The Content Linking component is registered with the ProgId of "`MSWC.NextLink`." The following VBScript excerpt creates an instance of the control.

```
Set cLinker = Server.CreateObject("MSWC.NextLink")
```

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*]]

Parameters: 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 the content path, make sure that no collisions or infinite loops can occur.

text-description

A value containing text that describes *Web-page-URL*.

comment

Explanatory text not processed by the component.

Example: 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 Baseball Playoffs
story2.htm New Health Initiative Passes
story3.htm Hot Recipes for Cool Nights
story4.htm Nice Weather on the Way
story5.htm Reducing Stress on the Job
main.htm Return to the table of contents
```

ASP Content Linking Component Properties

None

ASP Content Linking Component Methods

The Content Linking component provides the following methods:

- **GetListCount**
- **etListIndex**
- **GetNextDescription**
- **GetNextURL**
- **GetNthDescription**
- **GetNthURL**
- **GetPreviousDescription**

- **GetPreviousURL**

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.

Example: 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 need 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 retrieves or displays a new HTML content string based upon information specified 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 does not use registry settings.

Syntax: ASP Content Rotator Component

The Content Rotator component is registered with the ProgId of "`MSWC.ContentRotator`." The following VBScript creates an instance of the control.

```
Set NextTip = Server.CreateObject("MSWC.ContentRotator")
```

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

The Content Rotator component Content Schedule file parameters are listed below.

Weight

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 contains 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.

Comments

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 (`//`).

ContentString

This is 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 percentage signs (`%%`) as a single HTML content string.

Example: ASP Content Rotator Component Content Schedule File Parameters

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 Properties

None

ASP Content Rotator Component Methods

The Content Rotator component provides the following methods:

- **ChooseContent**
- **GetAllContent**

ASP Content Rotator Component ChooseContent Method

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 reside 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 "ASP Server Object MapPath Method" on page 258.

Return Value: ASP Content Rotator Component ChooseContent Method

Returns an HTML content string from the Content Schedule file.

Example: 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("/MyApp/Tips/Content.txt")
 Response.Write Tip
%>
```

ASP Content Rotator Component GetAllContent Method

The **GetAllContent** method retrieves all 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 as a relative or virtual path. For example, if the Content Schedule file, Content.txt, and the .asp file that called **GetAllContent** both reside 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 "ASP Server Object MapPath Method" on page 258.

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 "ASP Response Object Write Method" on page 250.

Examples: ASP Content Rotator Component GetAllContent Method

The following example uses the **GetAllContent** method to display all entries in the Content Schedule file.

```
<H1>Tips Stored in the Content Schedule File:</H1>
<%
 Set Tips = Server.CreateObject("MSWC.ContentRotator")
 Tips.GetAllContent("/MyApp/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 Counters 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 it is removed.

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, Counter.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 does not use 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>
```

ASP Counters Component Properties

None

ASP Counters Component Methods

The Counters component provides the following methods:

- **Get**
- **Increment**
- **Remove**
- **Set**

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 of 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 Counter.txt file.

Arguments: ASP Counters Component Remove Method

CounterName A string containing the name of the counter.

Example: ASP Counters Component Remove Method

The following code removes the counter `hitscounter` from the Counter.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.

Example: 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. The administrator typically types this information directly into the Web server interface. However, the values of the properties can be set directly by using a script in an ASP page.

Note

Sun ONE 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 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 (platform-specific), where [C-ASP_INSTALL_DIR] is the complete directory path of the Sun ONE ASP installation directory.

The Sun ONE 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 ONE ASP does not.

Registry Settings: ASP MyInfo Component

The MyInfo component does not use 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>
```

ASP MyInfo Component Properties

The Sun ONE implementation does not implement the default properties available under Windows Personal Web Services. You create your own properties as described in this section.

ASP MyInfo Component Methods

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 Tools component does not use registry settings.

Syntax: ASP Tools Component

The Tools component is registered with the ProgId of "MSWC.Tools." The following VBScript excerpt creates an instance of the control.

```
Set Tools = Server.CreateObject("MSWC.Tools")
```

The Tools component exposes the following properties and methods.

ASP Tools Component Properties

None

ASP Tools Component Methods

The Tools component provides the following methods:

- **FileExists**
- **Owner**
- **PluginExists**
- **ProcessForm**
- **Random**

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, and FALSE if the file does not exist.

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.

Example: 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 ONE ASP implementation of this method always returns False.

ASP Tools Component PluginExists Method

The Sun ONE ASP implementation of this method always returns False.

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 built-in objects for the page executing the **ProcessForm** method except 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 *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.

InsertionPoint is not supported in this release of Sun ONE ASP.

Example: 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")
%>
```

ASP Tools Component Random Method

The **Random** method returns an integer between –32768 and 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.

Example: ASP Tools Component Random Method

```
<% = Tools.Random %> displays a random integer between -32768 to 32767.
For example, -13067.
<% = ( Abs( Tools.Random ) ) %> displays a positive random integer.
For example, 23054.
<% = ( Abs( Tools.Random ) ) Mod 100 %> displays a random integer
between 0 and 99. For example, 63.
```

11 ADO Component Reference

Sun ONE Active Server Pages 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 chapter provides ADO reference information.

In this chapter:

ADO Overview

The implementation of ADO used with Sun ONE 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 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 these drivers, Sun uses the implementation of updatable cursors in the ODBC 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.

Note

ADO objects cannot be stored in application variables.

Object	Description
"ADO Command Object" on page 303	Defines a specific command to execute against a data source.
"ADO Connection Object" on page 318	Represents an open connection to a data source.
"ADO Error Object" on page 346	Provides specific details about each ADO error.
"ADO Field Object" on page 351	Represents a column of data with a common data type.
"ADO Parameter Object" on page 364	Represents a parameter or argument associated with a **Command** object based on a parameterized query or stored procedure.
"ADO Property Object" on page 373	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" on page 379	Represents the entire set of records from a database table or the results of an executed command.

ADO Command Object

The **Command** object defines a specific command to execute against a data source.

ADO Command Object Collections

Collection	Description
"ADO Parameters Collection" on page 455	Contains all the **Parameter** objects of a **Command** object.
"ADO Properties Collection" on page 456	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

Method	Description
"ADO Command Object CreateParameter Method" on page 304	Creates a new **Parameter** object with the specified properties.
"ADO Command Object Execute Method" on page 305	Executes the query, SQL statement, or stored procedure specified in the **CommandText** property.

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" on page 370 for valid settings.

Direction

An optional **Long** value specifying the type of **Parameter** object. See the "ADO Parameter Object Direction Property" on page 368 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" on page 456 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.

Constant	Description
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" on page 313 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")
Set 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

Property	Description
"ADO Command Object ActiveConnection Property" on page 310	The **Connection** object to which the specified **Command** object currently belongs.
"ADO Command Object CommandText Property" on page 312	The text of a command that you want to issue against a provider.
"ADO Command Object CommandTimeout Property" on page 313	How long to wait while executing a command before terminating the command and issuing an error.
"ADO Command Object CommandType Property" on page 313	The type of **Command** object.
"ADO Command Object Name Property" on page 314	The name of a specific **Command** object. *This property is not currently supported on UNIX*.
"ADO Command Object Prepared Property" on page 314	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" on page 316	The current state of the **Command** object. *This property is not currently supported on UNIX.*

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 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 Connection Object

A **Connection** object represents an open connection to a data source.

ADO Connection Object Collections

Collection	Description
"ADO Errors Collection" on page 454	Contains all stored **Error** objects that pertain to an ADO operation.
"ADO Properties Collection" on page 456	All **Property** objects for a specific instance of a **Connection** object. *This collection is not currently supported on UNIX.*

ADO Connection Object Methods

Method	Description
"ADO Connection Object Close Method" on page 318	Closes an open **Connection** object and any dependent objects.
"ADO Connection Object Execute Method" on page 322	Executes the specified query, SQL statement, stored procedure, or provider-specified text.
"ADO Connection Object Open Method" on page 327	Opens a connection to a data source.
"ADO Connection Object OpenSchema Method" on page 323	Obtains database schema information from the provider. *This method is not currently supported on UNIX.*
"ADO Connection Object BeginTrans, CommitTrans, and RollbackTrans Methods" on page 328	Cancels any changes made during the current transaction and ends the transaction. It may also start a new transaction.

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" on page 313 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" on page 305.

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 *Not currently supported on UNIX.*

QueryType values	Criteria values
adSchemaCatalogs	CATALOG_NAME *Not currently supported on UNIX*.
asSchemaCharacterSets	CHARACTER_SET_CATALOG CHARACTER_SET_SCHEMA CHARACTER_SET_NAME *Not currently supported on UNIX*.
adSchemaCheckConstraints	CONSTRAINT_CATALOG CONSTRAINT_SCHEMA CONSTRAINT_NAME *Not currently supported on UNIX*.
adSchemaCollations	COLLATION_CATALOG COLLATION_SCHEMA COLLATION_NAME *Not currently supported on UNIX*.
adSchemaColumnDomainUsage	DOMAIN_CATALOG DOMAIN_SCHEMA DOMAIN_NAME COLUMN_NAME *Not currently supported on UNIX*.
adSchemaColumnPrivileges	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME COLUMN_NAME GRANTOR GRANTEE *Not currently supported on UNIX*.
adSchemaColumns	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME **Supported on UNIX**.
adSchemaConstraintTableUsage	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME COLUMN_NAME *Not currently supported on UNIX*.
adSchemaForeignKeys	PK_TABLE_CATALOG PK_TABLE_SCHEMA PK_TABLE_NAME FK_TABLE_CATALOG FK_TABLE_SCHEMA FK_TABLE_NAME *Not currently supported on UNIX*.
adSchemaIndexes	TABLE_CATALOG TABLE_SCHEMA INDEX_NAME TYPE TABLE_NAME *Not currently supported on UNIX*.

QueryType values	Criteria values
adSchemaKeyColumnUsage	CONSTRAINT_CATALOG CONSTRAINT_SCHEMA CONSTRAINT_NAME TABLE_CATALOG TABLE_SCHEMA TABLE_NAME COLUMN_NAME *Not currently supported on UNIX*.
adSchemaPrimaryKeys	PK_TABLE_CATALOG PK_TABLE_SCHEMA PK_TABLE_NAME *Not currently supported on UNIX*.
adSchemaProcedureColumns	PROCEDURE_CATALOG PROCEDURE_SCHEMA PROCEDURE_NAME COLUMN_NAME *Not currently supported on UNIX*.
adSchemaProcedures	PROCEDURE_CATALOG PROCEDURE_SCHEMA PROCEDURE_NAME PARAMETER_TYPE *Not currently supported on UNIX*.
adSchemaProviderSpecific	see Remarks *Not currently supported on UNIX*.
adSchemaProviderTypes	DATA_TYPE BEST_MATCH **Supported on UNIX**.
adSchemaReferentialConstraints	CONSTRAINT_CATALOG CONSTRAINT_SCHEMA CONSTRAINT_NAME *Not currently supported on UNIX*.
adSchemaSchemata	CATALOG_NAME SCHEMA_NAME SCHEMA_OWNER *Not currently supported on UNIX*.
adSchemaSQLLanguages	<none> *Not currently supported on UNIX*.
adSchemaStatistics	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME *Not currently supported on UNIX*.

QueryType values	Criteria values
adSchemaTableConstraints	CONSTRAINT_CATALOG CONSTRAINT_SCHEMA CONSTRAINT_NAME TABLE_CATALOG TABLE_SCHEMA TABLE_NAME CONSTRAINT_TYPE *Not currently supported on UNIX*.
adSchemaTablePrivileges	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME *Not currently supported on UNIX*.
adSchemaTables	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME TABLE_TYPE **Supported on UNIX**.
adSchemaTranslations	TRANSLATION_CATALOG TRANSLATION_SCHEMA TRANSLATION_NAME *Not currently supported on UNIX*.
adSchemaUsagePrivileges	OBJECT_CATALOG OBJECT_SCHEMA OBJECT_NAME OBJECT_TYPE GRANTOR GRANTEE *Not currently supported on UNIX*.
adSchemaViewColumnUsage	VIEW_CATALOG VIEW_SCHEMA VIEW_NAME *Not currently supported on UNIX*.
adSchemaViewTableUsage	VIEW_CATALOG VIEW_SCHEMA VIEW_NAME *Not currently supported on UNIX*.
adSchemaViews	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME *Not currently supported on UNIX*.

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" on page 318.

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

Property	Description
"ADO Connection Object Attributes Property" on page 332	One or more characteristics of an object.

Property	Description
"ADO Connection Object CommandTimeout Property" on page 333	How long to wait while executing a command before terminating the command and issuing and error.
"ADO Connection Object ConnectionString Property" on page 334	Contains the information used to establish a connection to a data source.
"ADO Connection Object ConnectionTimeout Property" on page 336	How long to wait while establishing a connection before terminating the attempt and issuing and error.
"ADO Connection Object CursorLocation Property" on page 337	The location of the cursor engine in a recordset.
"ADO Connection Object DefaultDatabase Property" on page 338	The default database for the **Connection** object. *This property is not currently supported on UNIX*.
"ADO Connection Object IsolationLevel Property" on page 338	The level of isolation for the **Connection** object.
"ADO Connection Object Mode Property" on page 340	The available permissions for modifying data in a **Connection** object.
"ADO Connection Object Provider Property" on page 341	The name of a provider for a **Connection** object. *This property is not available on UNIX.*
"ADO Connection Object State Property" on page 342	Describes the current state of the **Connection** object.
"ADO Connection Object Version Property" on page 344	The ADO version number.

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.

Argument	Description
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 ONE 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 ONE 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. *This property is not currently supported on UNIX*.

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" on page 341

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.

Constant	Description
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 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 Error Object

The ADO **Error** object provides specific details about each ADO error.

ADO Error Object Properties

Note

None of the ADO **Error** object properties listed in this section are currently supported on UNIX.

Property	Description
"ADO Error Object Description Property" on page 346	A descriptive string associated with an error.
"ADO Error Object HelpContext, HelpFile Property" on page 348	The help file topic associated with an error.
"ADO Error Object HelpContext, HelpFile Property" on page 348	The help file associated with an error.
"ADO Error Object NativeError Property" on page 348	The provider-specific error code for an error.
"ADO Error Object Number Property" on page 349	The number that uniquely identifies an error.
"ADO Error Object Source Property" on page 349	The name of the object or application that originally generated the error.
"ADO Error Object SQLState Property" on page 350	The SQL state for a given error.

ADO Error Object Description Property

A descriptive string associated with an **Error** object. *This property is not currently supported on UNIX*.

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, HelpFile Property, 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. *This property is not currently supported on UNIX*.

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" on page 346 example.

ADO Error Object NativeError Property

The provider-specific error code for a given **Error** object. *This property is not currently supported on UNIX*.

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" on page 346.

ADO Error Object Number Property

The number that uniquely identifies an **Error** object. *This property is not currently supported on UNIX*.

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" on page 346.

ADO Error Object Source Property

The name of the object or application that originally generated an error. *This property is not currently supported on UNIX*.

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" on page 346 example.

ADO Error Object SQLState Property

The SQL state for a given **Error** object. *This property is not currently supported on UNIX*.

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" on page 346 example.

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, HelpFile Property 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 Field Object

The ADO Field Object represents a column of data with a common data type.

ADO Field Object Collections

"ADO Properties Collection" on page 456	All **Property** objects for a specific instance of a **Field** object. *This collection is not currently supported on UNIX.*

ADO Field Object Methods

Method	Description
"ADO Field Object AppendChunk Method" on page 352	Appends data to a large text or binary data field.
"ADO Field Object GetChunk Method" on page 352	Returns all or a portion of the contents of a large text or binary data field.

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" on page 352 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" on page 352.

ADO Field Object Properties

Property	Description
"ADO Field Object ActualSize Property" on page 354	The actual length of a field value.
"ADO Field Object Attributes Property" on page 355	One or more characteristics of a field. *This property is read-only on UNIX.*

Property	Description
"ADO Field Object DefinedSize Property" on page 356	The defined size of a field.
"ADO Field Object Name Property" on page 357	The name of a field.
"ADO Field Object NumericScale Property" on page 357	The scale of numeric values in a field.
"ADO Field Object OriginalValue Property" on page 357	The value of a field that existed in the record before any changes were made. *This property is not currently supported on UNIX*.
"ADO Field Object Precision Property" on page 359	The degree of precision for numeric values in a field.
"ADO Field Object Type Property" on page 360	The data type of the field.
"ADO Field Object UnderlyingValue Property" on page 362	The current value of the field in the database. *This property is not currently supported on UNIX*.
"ADO Field Object Value Property" on page 363	The value assigned to the field.

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.

Value	Description
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" on page 354 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" on page 355 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. *This property is not currently supported on UNIX*.

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" on page 357.

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).

Constant	Description
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. *This property is not currently supported on UNIX*.

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" on page 357 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 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 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" on page 456	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" on page 365	Appends data to a large text or binary data parameter.

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

Property	Description
"ADO Parameter Object Attributes Property" on page 366	One or more characteristics of a parameter. *This property is currently read-only on UNIX.*
"ADO Parameter Object Direction Property" on page 368	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" on page 368	The name of the parameter.
"ADO Parameter Object NumericScale Property" on page 369	The scale of numeric values in the parameter.
"ADO Parameter Object Precision Property" on page 369	The degree of precision for numeric values in the parameter.
"ADO Parameter Object Size Property" on page 369	The maximum size, in bytes or characters, of a parameter.
"ADO Parameter Object Type Property" on page 370	The data type of the parameter.
"ADO Parameter Object Value Property" on page 372	The value assigned to the parameter.

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.

Value	Description
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).

Constant	Description
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 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 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.*

ADO Property Object Properties

Property	Description
"ADO Property Object Attributes Property" on page 373	One or more characteristics of a property.
"ADO Property Object Name Property" on page 375	The name of the property.
"ADO Property Object Type Property" on page 375	The operational or data type of the property.
"ADO Property Object Value Property" on page 377	The value assigned to the property.

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.

Value	Description
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" on page 373 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.

Constant	Description
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).

Constant	Description
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 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 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

Collection	Description
"ADO Fields Collection" on page 455	All the stored **Field** objects of a **Recordset** object.
"ADO Properties Collection" on page 456	All the **Property** objects for a specific instance of a **Recordset** object. *This collection is not currently supported on UNIX.*

ADO Recordset Object Methods

Method	Description
"ADO Recordset Object AddNew Method" on page 380	Creates a new record for an updatable **Recordset** object.
"ADO Recordset Object CancelBatch Method" on page 381	Cancels a pending batch update. *This method is not currently supported on UNIX.*
"ADO Recordset Object CancelUpdate Method" on page 384	Cancels any changes made to the current record prior to calling the **Update** method.
"ADO Recordset Object Clone Method" on page 386	Creates a new **Recordset** object from an existing **Recordset** object. *This method is not currently supported on UNIX.*
"ADO Recordset Object Close Method" on page 387	Closes an open **Recordset** object and any dependent objects.
"ADO Recordset Object Delete Method" on page 391	Deletes the current record or group of records from a **Recordset** object.
"ADO Recordset Object GetRows Method" on page 394	Retrieves multiple rows from a **Recordset** object into an array.
"ADO Recordset Object Move Method" on page 397	Moves the position of the current record in a **Recordset** object.

Method	Description
"ADO Recordset Object MoveFirst, MoveLast, MoveNext, MovePrevious Methods" on page 402	Moves to the first, first, last, next record in a **Recordset** object and makes that record the current record.
"ADO Recordset Object NextRecordset Method" on page 406	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" on page 408	Opens a cursor.
"ADO Recordset Object Requery Method" on page 411	Updates the data in a recordset by re-executing the query on which the object is based.
"ADO Recordset Object Resync Method" on page 411	Refreshes the data in the **Recordset** object from the underlying database.
"ADO Recordset Object Supports Method" on page 413	Determines whether a specified **Recordset** object supports a particular type of functionality.
"ADO Recordset Object Update Method" on page 416	Saves any changes you make to the current record of a **Recordset** object.
"ADO Recordset Object UpdateBatch Method" on page 419	Writes all pending batch updates. *This method is not currently supported on UNIX.*

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, 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" on page 313 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" on page 387.

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" on page 305.

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" on page 381 example.

ADO Recordset Object Properties

Property	Description
"ADO Recordset Object AbsolutePage Property" on page 422	The page in which the current record resides.

Property	Description
"ADO Recordset Object AbsolutePosition Property" on page 424	The ordinal position of a **Recordset** object's current position.
"ADO Recordset Object ActiveConnection Property" on page 425	The **Connection** object to which the **Recordset** object currently belongs.
"ADO Recordset Object BOF, EOF Properties" on page 427	If BOF is **True**, the current record position is before the first record in a **Recordset** object. If EOF is **True**, the record position is after the last record in a **Recordset** object.
"ADO Recordset Object Bookmark Property" on page 430	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" on page 431	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" on page 433	The location of the cursor engine.
"ADO Recordset Object CursorType Property" on page 433	The type of cursor used in a **Recordset** object.
"ADO Recordset Object EditMode Property" on page 436	The editing status of the current record.
"ADO Recordset Object Filter Property" on page 437	A filter for data in a **Recordset** object.
"ADO Recordset Object LockType Property" on page 440	The type of locks placed on records during editing.
"ADO Recordset Object MarshalOptions Property" on page 441	Which records are to be marshaled back to the server.
"ADO Recordset Object MaxRecords Property" on page 443	The maximum number of records to return to a **Recordset** object from a query.
"ADO Recordset Object PageCount Property" on page 444	The number of pages of data the **Recordset** object contains.

Property	Description
"ADO Recordset Object PageSize Property" on page 444	The number of records that make up one page in the **Recordset** object.
"ADO Recordset Object RecordCount Property" on page 449	The current number of records in a **Recordset** object.
"ADO Recordset Object Source Property" on page 447	The source for the data in a **Recordset** object.
"ADO Recordset Object State Property" on page 445	Describes the current state of the **Recordset** object.
"ADO Recordset Object Status Property" on page 445	The status of the current record with respect to batch updates or other bulk operations.

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	**Move Previous Move < 0**	**Move 0**	**Move Next 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" on page 427.

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.

Cursor Type	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" on page 380 for a more detailed description of the **EditMode** property under different editing conditions.

EditMode Property Example

See the "ADO Recordset Object CursorType Property" on page 433 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" on page 430 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.

Constant	Description
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" on page 433 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)

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.

Constant	Description
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 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 Collections

Collections	Description
"ADO Errors Collection" on page 454	Contains all stored **Error** objects, all of which pertain to a single operation involving ADO.
"ADO Fields Collection" on page 455	Contains all stored **Field** objects of a **Recordset** object.
"ADO Parameters Collection" on page 455	Contains all the **Parameter** objects of a **Command** object.
"ADO Properties Collection" on page 456	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" on page 456	Appends a new object to the **Parameters** collection.

Collections	Description
"ADO Collections Clear Method" on page 458	Clears the contents of an **Errors** collection.
"ADO Collections Delete Method" on page 459	Deletes an object from the **Parameters** collection.
"ADO Collections Item Method" on page 459	Returns a specific member of a collection by name or ordinal number.
"ADO Collections Refresh Method" on page 460	Updates the objects in a collection to reflect objects available from and specific to the provider.
Properties	
"ADO Collections Count Property" on page 463	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" on page 346 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" on page 351 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" on page 373 topic for a more detailed explanation of how to use **Property** objects.

ADO Collections Methods

This section discusses 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" on page 305.

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

This section lists 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
```

12 Chili!Beans Component Reference

The Sun ONE ASP Chili!Beans ActiveX control is a wrapper that enables Java™ objects to be used by COM controllers (such as ActiveX scripting engines like VBScript). The control is designed to work with Java virtual machine (JVM™) versions 1.4 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 Sun ONE ASP Administration Console. JRE 1.4 is installed with Sun ONE ASP by default, and is required for use with the product.

Chili!Beans technology is also used to implement the ASP servlet interface new to this release of Sun ONE ASP. Java components designed for use in servlets and JSPs (Java Server Pages™) can now be integrated into Sun ONE Active Server Pages applications directly from ASP scripting (see "ASP Servlet Interface" on page 472 for more information).

Please note the following:

- When using Chili!Beans with Sun ONE 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" on page 469 in this section.

This chapter provides Chili!Beans reference information.

In this chapter:

Enabling Chili!Beans

Chili!Beans is enabled from the **Components** page in the Sun ONE ASP Administration Console. To use Chili!Beans, a Java runtime environment (JRE) must be installed on the machine, and Chili!Beans must be enabled in the Administration Console. JRE 1.4 is installed with Sun ONE ASP by default, and is required for use with the product.

When Chili!Beans is enabled you have the option to enable or disable the Java virtual machine (VM) Security Manager (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.

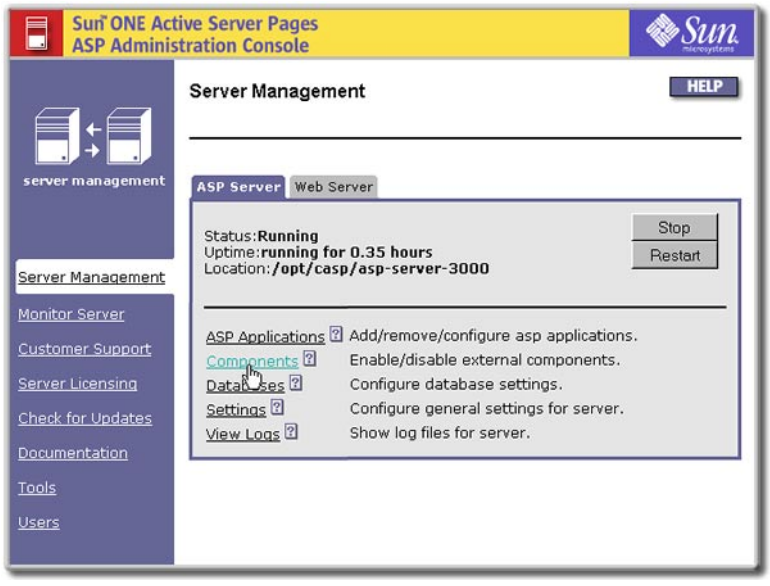
Note

For security reasons, the Java VM Security Manager should be enabled in multi-user environments in which users supply their own Java classes.

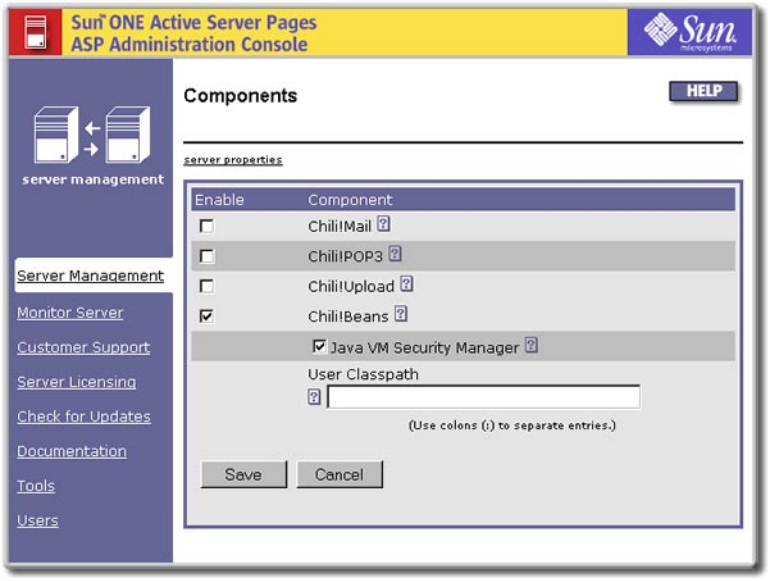
To selectively grant other privileges to Java code running in the Chili!Bean, with Java VM Security Manager enabled, use policytool to change the virtual machine's security settings as specified in the Java 2 Security documentation.

To enable or disable Chili!Beans

1. Open the Administration Console (see "Accessing the Administration Console" on page 18).
2. On the **ASP Server** tab of the **Server Management** page, 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 and is selected by default. Select or clear this box to enable or disable the Java VM Security Manager. The **User Classpath** field also displays. Use this field to specify a classpath other than the default. Paths can be separated by semicolons (;).

4. Click **Save** to save your changes, or **Cancel** to revert to the settings that were last saved.

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 (the CB_STACKTRACE setting is in javasetup.sh). With Sun ONE ASP as the controller, this string is reported as part of the run-time error text and will appear in the browser.

Note

The Chili!Beans control cannot be used to access static methods and fields.

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 ONE 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.
- The Chili!Beans object does not convert Java arrays, which are elements of other arrays, to script arrays. There is no support for multi-dimensional arrays as either arguments or return values. The Chili!Bean converts objects to strings by calling the Java **toString** method. The value displayed for the ARRAY_INT object is the value returned by the **toString** method.

Supplying Java Virtual Machine Settings

In rare cases you must supply startup settings to the Java virtual machine (JVM). In a stand-alone Java application, these settings are passed as command-line arguments.

Note

The mechanism described here is for expert 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:

<C-ASP_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 for more information.

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 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 several ways to create a Java object by using Chili!Beans, as discussed in this section. The recommended (and simplest) way is to use the **NewJavaObject** method (see "Accessing a Java Class via Chili!Beans" on page 470).

Accessing a Java Class via Chili!Beans

Sun ONE ASP 4.0 simplifies the creation of Java objects by providing a new Chili!Beans object method that allows you to instantiate Java classes with a single line of code. The new method is **NewJavaObject**. With this method, instantiating the Chili!Beans object and then calling the **Construct** method are combined into one step, so:

```
Set y = NewJavaObject("foo.bar")
```

has the same effect as

```
Set z = Server.CreateObject("Chili.Beans")
Set y = z.Construct("foo.bar")
```

While use of **NewJavaObject** is recommended, you can also use the Chili!Beans object **Construct** method to create instances of Chili!Beans. With the **Construct** method, 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 script.

To use a Java class with Sun ONE 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 ONE ASP as described in "Registering a Java Class as a COM Component on Linux and UNIX" on page 470 in this section. A classpath can also be specified via the Sun ONE ASP Administration Console, as described in "Enabling Chili!Beans" on page 466.

Caution

Static variables in a Java class are shared by every user who creates an instance of that class. This can create unexpected behavior because conflicts can occur when two instances of a class (even in different scripts) attempt to change the value.

Registering a Java Class as a COM Component on Linux and UNIX

The chregclass tool included with Sun ONE 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.

Note

To register a Java class to use with Sun ONE 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 ONE ASP installation directory.
2. Stop the ASP Server, as described in "Stopping and Restarting the ASP Server (Admin Console)" on page 41.
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 (Admin Console)" on page 41.

For example, to register the **Table** class in the **Database** package 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 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 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, int 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 = NewJavaObject("Database/Table", "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.

ASP Servlet Interface

Sun ONE Active Server Pages 4.0 implements some of the interfaces and classes in the **javax.servlet** and **javax.servlet.http** packages. This means that Java objects designed for use in Java Server Pages (JSPs) can now be integrated into a Sun ONE ASP script. The ASP servlet interface implemented in this release is not a full-fledged servlet container, but instead provides a mapping between servlet container objects and ASP objects (as described in "Object Mapping" on page 473).

Sun ONE ASP Chili!Beans technology is used to implement this functionality. Chili!Beans now has the ability to interact with Java classes that use the **ServletContext** interface. Java methods taking **HttpServletRequest** and **HttpServletResponse** arguments can be called directly from ASP scripts through the Chili!Beans wrapper. When methods of classes wrapped by Chili!Beans require arguments that implement these interfaces, Chili!Beans provides a mechanism that allows the methods to be called using a syntax similar to method calls from Java or JSP code.

This section provides information specific to the ASP servlet interface implemented in Sun ONE ASP. It does not provide a complete developer reference for the **javax.servlet** and **javax.servlet.http** interfaces, or for servlet or JSP technologies in general. The following URLs are good resources for that information:

- javax.servlet and javax.servlet.http packages

  http://java.sun.com/j2ee/1.4/docs/api/
- Java™ Servlet technology

  http://java.sun.com/products/servlet/
- JavaServer Pages™ technology

http://java.sun.com/products/jsp/

- Glossary of Java-related terms

  http://java.sun.com/docs/glossary.html

Object Mapping

The ASP servlet interface is a Java package with classes that map to servlet interfaces. The package is named **com.sun.asp**. The ASP servlet interface is not a full-fledged servlet container, but instead provides a mapping between servlet container objects and ASP objects. The following table illustrates that mapping.

Servlet Container Object	ASP Intrinsic Object
HttpServletContext	**Application**
HttpServletRequest	**Request**
HttpServletResponse	**Response**
HttpSession	**Session**

Calls made to the servlet container object are handled by the equivalent ASP object (if possible). If the functionality is not supported in ASP, an exception is thrown or the appropriate error code is returned. This mapping allows servlet methods that depend on servlet container objects to continue to function when called via Chili!Beans. However, the servlet will receive no calls to its **Listener** methods from the ASP Server.

See also:

"Programmatic Access" on page 473

Programmatic Access

The Java wrappers for the ASP intrinsic objects have two uses:

- The wrappers are used to call methods of existing Java classes designed to be called from servlets and JSPs. Such methods often take **Servlet** interface arguments. These methods can now be called directly from ASP by substituting the ASP **Request** and **Response** objects for the **ServletRequest** and **ServletResponse** arguments to these methods. This mechanism is illustrated in the first method of the **AspTest.asp** portion of the following example.
- Developers can write Java methods intended for use by ASP pages and access the ASP intrinsic objects directly from Java code. This mechanism is illustrated in the second and third methods in the **AspTest.asp** portion of the following example.

  A class called **com.sun.asp.AspContext** implements **javax.servlet.ServletContext** on the CLASSPATH of the virtual machine running in the ASP

engine. This class has a static method called **getScriptingContext**, which returns an instance of **AspContext** wrapping the ASP intrinsic objects for the current page. In addition to the **ServletContext** methods, this class has **getResponse()** and **getRequest()** methods returning **HttpServletRequest** and **HttpServletResponse** wrapping the current ASP **Request** and **Response** objects. (The **com.sun.asp.AspContext** class has no use outside of ASP, and will not work outside of ASP.)

Example

The following example illustrates ASP servlet functionality. Running **AspTest.asp** with the compiled **AspTest.class** on the classpath will display the contents of **AspTest.asp.html** when the request `/AspTest.asp?foo=bar` is sent to the browser.

AspTest.asp

```
<%
Set obj = NewJavaObject("AspTest")

'Display the query string using the first method. This Java method
'takes HttpServletRequest and HttpServletResponse arguments, so
'the ASP Response and Request objects are passed.
obj.FirstTest Request, Response

Response.write "<br><br>"
'Display the query string using the second method. This Java method
'accesses the ASP Request and Response objects directly from the
'Java code.
obj.SecondTest

Response.Write "<br><br>"
'Set a session variable from ASP and display it from Java.
Session("key") = "value"
obj.ThirdTest

%>
```

AspTest.class

```
import javax.servlet.*;
import javax.servlet.http.*;
import com.sun.asp.*;
import java.io.IOException;
```

```
import java.io.PrintWriter;

public class AspTest {
    public AspTest(){}

    //This method can be called from ASP by passing the ASP Request
    //and Response objects as arguments. The Chili!Bean recognizes
    //those objects and substitutes instances of its own
    //implementations of HttpServletRequest and
    //HttpServletResponse.
    public void FirstTest(HttpServletRequest req,
    HttpServletResponse res)
       throws IOException {
       PrintWriter writer = res.getWriter();
       String queryString = req.getQueryString();

       writer.println("The QueryString is:<br>");
       writer.println(queryString);
    }

    //This method does the same thing. In this case,
    //instances of HttpServletRequest and HttpServletResponse
    //that wrap the ASP Request and Response objects are
    //obtained in the Java code by calling the AspContext static
    //method getScriptingContext. The returned AspContext
    //object has methods that return the Request and
    //Response wrapper.
    public void SecondTest()
       throws IOException {
       AspContext ctxt = AspContext.getScriptingContext();
       HttpServletRequest req = ctxt.getRequest();
       HttpServletResponse res = ctxt.getResponse();
       FirstTest(req,res);
    }

    //This method displays the value of a session variable.
    //The Java code accesses the session data from the ASP
    //page calling it.
    public void ThirdTest()
```

```
        throws IOException {
        AspContext ctxt = AspContext.getScriptingContext();
                HttpServletRequest req = ctxt.getRequest();
                HttpServletResponse res = ctxt.getResponse();
        HttpSession sess = req.getSession();
        PrintWriter writer = res.getWriter();

        writer.println("The value of the session variable key is " +
           sess.getAttribute("key") + "<br>");
    }

}
```

AspTest.asp.html

```
<html><head></head><body>The QueryString is:<br>
foo=bar
<br><br>The QueryString is:<br>
foo=bar
<br><br>The value of the session variable key if
value<br></body></html>
```

Functionality Not Implemented

Not all servlet functionality has been implemented in this release of Sun ONE ASP. Functionality that has NOT been implemented is listed in the following sections. Complete reference information for the **javax.servlet** and **javax.servlet.http** interfaces can be found at the following URL:

http://java.sun.com/j2ee/1.4/docs/api/

ServletContext

Class **AspServletContext** implements Interface **javax.servlet.ServletContext**. **ServletContext** functionality not implemented in **AspServletContext**:

- **getMimeType**
- **getResourcePaths**

Note

Request dispatchers are not supported in this release of Sun ONE ASP.

HttpServletRequest

Class **AspServletRequest** implements Interface **javax.servlet.http.HttpServletRequest**. **HttpServletRequest** functionality not implemented in **AspServletRequest**:

- **getRequestedSessionId**
- **getServletPath**
- **getUserPrincipal**
- **isRequestedSessionIdValid**
- **isUserInRole**

HttpServletResponse

Class **AspServletResponse** implements Interface **javax.servlet.http.HttpServletResponse**. **HttpServletResponse** functionality not implemented in **AspServletResponse**:

- **containsHeader**

Methods Defined in javax.servlet.ServletResponse

Methods defined in **javax.servlet.ServletResponse** that are not implemented:

- **reset**

HttpSession

Class **AspSession** implements interface **javax.servlet.http.HttpSession**. **HttpSession** functionality not implemented in **AspSession**:

- **getCreationTime**
- **getCreationTime**
- **getLastAccessedTime**
- **isNew**

13 XML Support

Sun ONE Active Server Pages provides built-in support for XML, enabling developers to incorporate pages using the Document Object Model (DOM) and HTTP features of MSXML 1.0 into their Sun ONE ASP applications with few changes to code. (MSXML 1.0 is a very basic DOM-based XML parser. Support for DOM Level 2 is provided.)

The Sun ONE ASP XML control is based on the Java API for XML Processing (Java XML) implementation. An MSXML 1.0 DOM compatibility layer allows syntax written for use with the Microsoft control to function with minimal modification. The Sun ONE ASP XML control extends the **org.w3c.dom** interfaces, which are the interfaces for the DOM that is a component API of the Java XML implementation.

This chapter lists MSXML 1.0 DOM extensions to the W3C DOM that are NOT implemented in the Sun ONE ASP control. It does not provide a complete developer reference for the MSXML 1.0 DOM and the **org.w3c.dom** package or for Java XML, XML, and so on. Many other resources provide that information, including the following:

- org.w3c.dom package

  http://java.sun.com/webservices/docs/1.0/api/
- Java XML

  http://java.sun.com/xml/
- MSXML

  MSDN resources
- DOM

  http://www.w3.org/DOM/

In this chapter:

About the Sun ONE ASP XML Control

The Sun ONE ASP XML control is a Java package with interfaces that correspond to COM interfaces in the MSXML 1.0 DOM implementation. The package is named **com.sun.msxml** and does not need to be registered. The following is provided for each interface in the Microsoft DOM 1.0 implementation:

- A corresponding interface in the **com.sun.msxml** package that extends an **org.w3c.dom** interface and may include Microsoft-specific methods not found in the base interfaces.

- An implementation of each interface in the Microsoft package. Not all Microsoft-specific properties and methods are implemented in the Sun ONE ASP XML control. Some methods not contained in **org.w3c.dom** may throw FeatureNotImplemented exceptions. Others may silently fail, allowing ASP pages that invoke them to run without error. The behavior of each interface member with a corresponding method in the W3C specification for DOM parsers is determined by the W3C specification, without regard to the behavior of the Microsoft DOM parser. The behavior of Microsoft-specific methods corresponds as closely as possible to the behavior of the methods in the Microsoft software.

See also

"Functionality Not Implemented" on page 480

Functionality Not Implemented

Most but not all of the MSXML 1.0 DOM properties and methods are implemented in the Sun ONE ASP XML control. Functionality that has NOT been implemented in this release is listed in the following sections. The properties and methods added to MSXML since the release of MSXML 1.0 are not implemented.

Node Interface

The following table lists the MSXML 1.0 DOM functionality not implemented in the Sun ONE ASP XML control.

MSXML 1.0 control	Sun ONE ASP XML control
dataType	Not implemented. Throws FeatureNotImplemented exception.
definition	Not implemented. Throws FeatureNotImplemented exception.

Document Interface

The following table lists the MSXML 1.0 DOM functionality not implemented in the Sun ONE ASP XML control.

MSXML 1.0 control	Sun ONE ASP XML control
async	Not implemented. Throws FeatureNotImplemented exception.
abort	Not implemented. Throws FeatureNotImplemented exception.

MSXML 1.0 control	Sun ONE ASP XML control
ondataavailable	Not implemented. Throws FeatureNotImplemented exception.
onreadystatechange	Not implemented. Throws FeatureNotImplemented exception.
ontransformnode	Not implemented. Throws FeatureNotImplemented exception.

XMLHTTPRequest Object

The following table lists the MSXML 1.0 DOM functionality not implemented in the Sun ONE ASP XML control.

MSXML 1.0 control	Sun ONE ASP XML control
onreadystatechange	Not implemented. Throws FeatureNotImplemented exception.
readyState	Not implemented. Throws FeatureNotImplemented exception.
responseStream	Not implemented. Throws FeatureNotImplemented exception.

14 SpicePack Component Reference

The Sun ONE ASP 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 ONE Active Server Pages and are enabled or disabled from the Sun ONE ASP Administration Console. This chapter provides SpicePack reference information.

In this chapter:

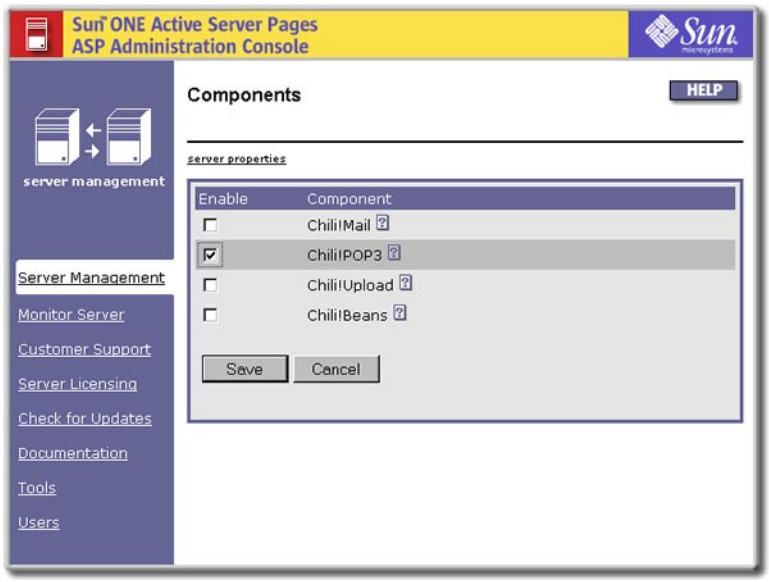
Enabling SpicePack Components

The SpicePack components are enabled or disabled from the Sun ONE Active Server Pages Administration Console.

To enable or disable SpicePack components

1. If necessary, open the Administration Console (see "Accessing the Administration Console" on page 18).
2. On the **ASP Server** tab of the **Server Management** page, click **Components**.

   The **Components** page displays.



3. Select or clear (enable or disable) the **Chili!Mail**, **Chili!POP3**, and **Chili!Upload** check boxes as desired.

   If **Chili!Upload** is selected, the **Max. Transfer Size (Bytes)** box displays. This box specifies the maximum size per transfer (in bytes) that can be uploaded using the Chili!Upload component.

4. Click **Save** to save your changes, or **Cancel** to revert to the settings that were last saved.

5. If you changed the status of the Chili!Upload 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 Chili!Mail or Chili!POP3 components.

Note

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

See also:

"SpicePack Component Reference" on page 483

"Chili!Mail (SMTP)" on page 484

"Chili!POP3 (POP3)" on page 491

"Chili!Upload (File Upload)" on page 499

Chili!Mail (SMTP)

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**

Any differences between the Microsoft **NewMail** object and the Chili!Mail component are listed in the property and method descriptions that follow.

Note

The Chili!Mail component must be enabled in the Sun ONE ASP Administration Console. For more information, see "Enabling SpicePack Components" on page 483.

In this section:

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**
- **Charset**
- **Codepage**
- **From**
- **Host**
- **Importance**
- **Retain**
- **Subject**
- **To**
- **Value**
- **WrapLength**

Note

The **To**, **From**, **Subject**, and **Body** Chili!Mail properties can be set as arrays of bytes obtained from **Request.BinaryRead** or method calls on other objects. If one of the string properties is set in this manner, the translation of bytes in the array to Unicode is not performed. The values of these strings can be set to the exact bytes ultimately sent to the mail server, which allows multibyte text not encoded using a supported code page to be sent by the Chili!Mail component.

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 (the 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 Charset Property (String: Read/Write)

This property enhances functionality of the Chili!Mail component with non-Roman character sets, enabling a specific character set to be specified for the e-mail message being sent.

The **Charset** property specifies the value to be written in the Content-Type header of the MIME part containing the message body. The default is "ISO-8559-1" if the message contains high ASCII characters, otherwise it's "US-ASCII." In each header that contains non-ASCII characters, the value is written in accordance with RFC 2047.

No validation is done to verify that the value of the **Charset** property as set represents a valid or recognizable character set. The **Charset** property has no effect on the code page used for Unicode/multibyte conversion.

Chili!Mail CodePage Property (Integer: Read/Write)

This property enhances the functionality of the Chili!Mail component with non-Roman character sets, enabling a specific code page to be used for a single message.

By default, all e-mail messages and headers are encoded using the code page that is currently in effect. In some cases, however, you may want to use a specific code page for just a single message.

If set, the value of the **CodePage** property defines the code page used to convert the Unicode strings passed to the mail server. The default is the code page currently in effect, which is determined in the following order:

1. If **Response.CodePage** is set in the page, then the current value of **Response.CodePage** is the effective code page.
2. If **Session.CodePage** is set in the page, then the current value of **Session.CodePage** is the effective code page.
3. If the page contains an @CODEPAGE directive:

   <%@ **CODEPAGE**=*codepage* %>

   then *codepage* is the effective code page.
4. If none of the previous conditions exist, the setting in casp.cnfg is the effective code page (configurable from the Sun ONE ASP Administration Console).

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 approach is to limit the string to alphanumeric characters, dashes, and slashes, and in particular to avoid spaces.

The **Value** property can be set 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 (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 990.

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.

Chili!Mail AttachFile Method Arguments

Source A string containing the absolute path name of the file to attach.

Note

This note pertains to CDONTS. 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

The arguments listed in the following table must be passed in the order given.

From	See the description of the property of the same name: "Chili!Mail From Property (String: Read/Write)" on page 488.
To	See the description of the property of the same name: "Chili!Mail To Property (String: Read/Write)" on page 488.
Subject	See the description of the property of the same name: "Chili!Mail Subject Property (String: Read/Write)" on page 488.
Body	See the description of the property of the same name: "Chili!Mail Body Property (String: Read/Write)" on page 486.
Importance	See the description of the property of the same name: "Chili!Mail Importance Property (Long: Read/Write)" on page 488.
Host	See the description of the property of the same name: "Chili!Mail Host Property (String: Read/Write)" on page 488.

Chili!Mail Send Method Examples

Example 1

```
Set mailmsg = Server.CreateObject("CDONTS.NewMail")
mailmsg.To = "youraccount@yourco.com"
mailmsg.From = "account@someco.com"
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)

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, and the Message interface exposes all properties of a single message. Additional interfaces are exposed to support retrieval of message lists and message attachments.

Note

The Chili!POP3 component must be enabled in the Sun ONE ASP Administration Console. For more information, see "Enabling SpicePack Components" on page 483.

In this section:

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 in "Chili!POP3 Message Interface" on page 493. 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**
- **Disconnect**
- **Delete**
- **Reset**

POP3 Interface Connect Method

The **Connect** method establishes a network connection to a POP3 server.

POP3 Interface Connect Method Arguments

Host	The hostname of the server with which to connect.
Password	The password required for connecting with the server.
UserId	The User ID required for connecting with the server.

POP3 Interface Connect Method Example

See "POP3 Interface Disconnect Method" on page 492

POP3 Interface Disconnect Method

The **Disconnect** method disconnects from the POP3 server.

POP3 Interface Disconnect Method Example

```
Set pop3 = Server.CreateObject("Chili.Pop3.1")
pop3.Connect "mail.foo.com", "myuserid", "mypasswd"
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.

POP3 Interface Delete Method Arguments

Id	0-based index for the message in the message collection.

POP3 Interface Reset Method

The **Reset** method 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 undeleted state.

POP3 Interface Reset Method Example

```
Set pop3 = Server.CreateObject("Chili.Pop3.1")
pop3.Connect "mail.foo.com", "myuserid", "mypasswd"
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

In the list of properties below, LW means it can be "lightweight" on POP3 servers supporting the TOP command.

- **Charset** (LW)
- **DateReceived** (LW)
- **DateSent** (LW)
- **From** (LW)
- **HasAttachments** (LW)
- **Message**
- **MsgId** (LW)
- **MsgUID** (LW)
- **Size** (LW)
- **Subject** (LW)

Message Interface Charset Property (Read-Only)

After reading the value of **Charset**, you can set the value of **Session.CodePage** to correspond to the charset. That code page will be used to convert the message body and headers received from the mail server to Unicode strings returned by the object's properties.

Message Interface DateReceived Property (Read-Only)

The **DateReceived** property indicates the date and time that the message was received.

Message Interface DateSent Property (Read-Only)

The **DateSent** property indicates the date and time that the message was sent.

Message Interface From Property (Read-Only)

The **From** property is a string that indicates who sent the e-mail message.

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 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 Size Property (Read-Only)

The **Size** property indicates the total size of the current message in bytes.

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 Collections

The Message interface collections are as follows:

- **Attachments**
- **Cc** (LW)
- **Headers** (LW)
- **To** (LW)

Note

To, **Cc**, 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** and **Cc**, 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**, and **Subject**).

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 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 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 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 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 with 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.

Message Interface SaveAttachments Example

```
Set pop3 = Server.CreateObject("Chili.Pop3.1")
pop3.Connect "mail.foo.com", "myuserid", "mypasswd"
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

- **Base64**
- **ContentType**
- **FileName**
- **FileSize**

Attachment Interface Base64 Property (Read-Only)

The **Base64** property is a Boolean value that indicates whether the attachment is Base64 encoded.

Attachment Interface ContentType Property (Read-Only)

The **ContentType** property is a string that indicates the content type of the attachment.

Attachment Interface FileName Property (Read-Only)

The **FileName** property is a string that indicates the name 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 Methods

- **Read**
- **SaveToFile**

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.

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.

Chili!Upload (File Upload)

The Chili!Upload component enables users to save files uploaded by site visitors to the server. Chili!Upload supports the simultaneous upload of multiple files. Form data can also be processed while files are uploading.

Note

The Chili!Upload component must be enabled in the Sun ONE ASP Administration Console. For more information, see "Enabling SpicePack Components" on page 483.

In this section:

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 Upload = Server.CreateObject("Chili.Upload.1")
```

Chili!Upload Properties

The Chili!Upload component exposes the properties listed below. If the uploaded file contains multiple files, the properties of the first file being uploaded are accessed.

- **AllowOverwrite**
- **FileSize**
- **SizeLimit**
- **SourceFileExtension**
- **Version**

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 FileSize Property (Read-Only)

The **FileSize** property indicates the size in bytes for the uploaded file.

Chili!Upload SizeLimit Property (Read/Write)

The **SizeLimit** property sets the maximum file size in bytes of uploaded files.

Chili!Upload SourceFileExtension Property (Read-Only)

The **SourceFileExtension** property indicates the file extension of the uploaded file.

Chili!Upload Version Property (Read-Only)

The **Version** property indicates the version of the Chili!Upload component.

Chili!Upload Collections

The Chili!Upload component has the following collection:

- **FormData**

Chili!Upload FormData Collection

The **FormData** collection is a collection of **FormElement** objects that represent all input items on an HTML form.

Note

Every input element in the body of the HTTP request is included in the **FormData** collection, including empty File form fields.

FormElement Object

FormElement object properties are listed in the table below. All properties are read-only.

Property	Description
ContentType	String for the content type of the input item (if the item is not a file, the string will be empty).

Property	Description
FileName	String for the name of the file (if the item is not a file, the string will be empty). Also takes an optional Boolean argument. If FALSE, only the file name is returned. If TRUE (the default), the full path is returned (see "Chili!Upload SourceFileName Method (Read-Only)" on page 501).
FileNameExt	String for the file extension (if the item is not a file, the string will be empty).
IsFile	Boolean value to determine if the item is a file.
Name	String for the name of the input item in the form.
Size	Integer for the size of the item, in bytes.
Value	Safearray of bytes for the value of the input item.

Chili!Upload Methods

The Chili!Upload component exposes the following methods:

- **SaveToFile**
- **SourceFileName**

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.

Chili!Upload SourceFileName Method (Read-Only)

The **SourceFileName** method returns the full path or file name of the uploaded file. This method takes an optional Boolean argument. If `FALSE`, only the file name is returned. If `TRUE` (the default), the full path is returned.

Chili!Upload Methods Examples

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%>)
```

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 illustrates saving 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%>)
```

15 Scripting Languages Reference

Sun ONE Active Server Pages includes support for version 5.5 of Microsoft VBScript and JScript. Sun ONE ASP includes its own scripting engines, Sun ONE ASP VBScript and Sun ONE ASP JavaScript. Most functionality provided in Sun ONE ASP is equivalent to version 5.5 of Microsoft VBScript and JScript, including error messaging. Therefore, this section does not provide complete VBScript and JScript language references, but instead provides pointers to that information.

Differences between the Sun and Microsoft implementations do exist, however, and are documented in the README file included with Sun ONE ASP. The README file is the primary reference for this information.

Sun ONE ASP VBScript Reference

Sun ONE Active Server Pages includes the Sun ONE ASP VBScript scripting engine. Most but not all of the functionality is equivalent to version 5.5 of Microsoft VBScript.

- For Microsoft VBScript language reference information, including error messaging, go to:

  VBScript Language Reference (MSDN)

- For a list of differences between the Sun and Microsoft implementations, see the README file included with Sun ONE ASP. For information about accessing the README, see "Viewing the README File" on page 24.

Sun ONE ASP JavaScript Reference

Sun ONE Active Server Pages includes the Sun ONE ASP JavaScript scripting engine. Most but not all of the functionality is equivalent to version 5.5 of Microsoft JScript.

- For Microsoft JScript language reference information, including error messasging, go to:

  JScript Language Reference (MSDN)

- For a list of differences between the Sun and Microsoft implementations, see the README file included with Sun ONE ASP. For information about accessing the README, see "Viewing the README File" on page 24.

A Errors Reference

This appendix explains the error messages you might encounter when using Sun ONE Active Server Pages.

In this appendix:

Sun ONE ASP Errors

The following table describes Sun ONE ASP error messages, listing the error code, the error name, and an explanation of the error.

Error Code	Error Name	Explanation
100	Out of memory	Unable to allocate required memory.
101	Unexpected error	The function returned an exception.
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 an 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.

Error Code	Error Name	Explanation
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.

Error Code	Error Name	Explanation
132	Compilation error	The Active Server Page could not be processed.
133	Invalid ClassID attribute	The object tag has an invalid ClassID attribute.
134	Invalid ProgID attribute	The object has an invalid ProgID attribute.
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.

Error Code	Error Name	Explanation
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.

Error Code	Error Name	Explanation
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.

Error Code	Error Name	Explanation
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.

Sun ONE ASP VBScript Errors

See "Sun ONE ASP VBScript Reference" on page 503

Sun ONE ASP JavaScript Errors

See "Sun ONE ASP JavaScript Reference" on page 503

ADO Errors

The following table describes ADO error messages, listing the constant name, the number, and a description of the error.

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.

Constant Name	Number	Description
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.

B Troubleshooting

The Sun ONE Active Server Pages knowledge base provides troubleshooting information for problems you might encounter when using Sun ONE ASP.

The knowledge base is a valuable technical resource, providing an updated list of product-related articles, answers to frequently asked questions, and useful tips designed to help you get the most out of Sun ONE ASP.

To access the knowledge base, go to:

http://developer.chilisoft.com/kb/

See also:

"Other Resources" on page 13

C Advanced Administration Options

This appendix describes advanced administration options, and is designed for expert users of Sun ONE Active Server Pages.

Caution

Take great care when making the changes described in this section. Changes you make could require a complete reinstall of Sun ONE Active Server Pages and could void your eligibility for customer support. 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 ONE ASP Administration Console. It is strongly recommended that you use the Administration Console whenever possible. For more information, see "Chapter 2, Using the Administration Console" on page 17. Expert users can also perform certain management tasks from the command line, as described in "Chapter 5, Command-line Management" on page 83.

In this appendix:

Editing the Windows Registry

Sun ONE Active Server Pages for Windows stores some configuration information in the system registry. The registry settings used by Sun ONE ASP are listed in the table below, which provides the key, the default value, and a description. You can use regedit to edit these settings; regedit 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 ONE ASP and could void

your eligibility for customer support. Back up your data before making any changes.

Key	Default Values	Description
AllowOutOfProcCmpnts	0 (False)	Controls whether Sun ONE ASP allows Active Server Components that do not run in the Sun ONE 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 JavaScript. 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 ONE Active Server Pages has been disabled and cannot process your request."
EnableParentPaths	0 (False)	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. EnableParentPaths = False is the most secure setting and is appropriate for most shared Web hosting environments. Changing EnableParentPaths to True can affect the security of your server.
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.

Key	Default Values	Description
LogErrors	0 (False)	Determines if ASP Server errors should be written to the Sun ONE ASP log file.
LogToFile	0 (False)	This registry entry is set internally by the Sun ONE ASP engine to control logging of debug information. *Do not modify this setting.*
MaxThreads	10	This value controls the maximum number of threads per CPU that the Sun ONE ASP engine uses to process requests.
Running	00000001	This registry entry is set internally by the Sun ONE ASP engine to indicate whether Sun ONE ASP is running. *Do not modify this setting.*
ScriptEngineCacheMax	ffffffff	This value controls the maximum number of script engines that Sun ONE 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 ONE ASP Configuration File

UNIX and Linux versions of Sun ONE Active Server Pages include a configuration file, casp.cnfg, in which expert users can change Sun ONE ASP settings. This section describes the settings and their parameters. Most of the configuration settings described in this section are easily accessed from the Sun ONE ASP Administration Console. It is strongly recommended that you use the Administration Console whenever possible. For more information about changing server settings using the Sun ONE ASP Administration Console, see "Changing ASP Server Settings" on page 37.

Caution

Take great care when making the changes described in this section. Changes you make could require a complete reinstall of Sun ONE ASP and could void your eligibility for customer support. Back up your data before making any changes.

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 ONE 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 desired changes. For the changes to take effect, you must restart the ASP Server, as described in "Stopping and Restarting the ASP Server (Admin Console)" on page 41 or "Stop/Start/Status ASP Server (Command Line)" on page 84.

Note

If you make any changes to casp.cnfg, you must restart the Sun ONE ASP Server. If you make any changes to the `[default application]` section in casp.cnfg, you must restart both the Sun ONE ASP Server and the Web server.

The casp.cnfg file is divided into sections by keywords. The keywords and parameters are described in the following sections.

[machines]

The `[machines]` keyword defines the computers that are running the Sun ONE ASP Server. Parameters listed in the following table affect all Sun ONE ASP Servers.

Parameter	Explanation
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.
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 service requests.
disablerestart	This setting is useful for Sun ONE ASP diagnostics. If set to 1, the Sun ONE ASP parent process does not automatically re-spawn Sun ONE ASP child processes that fail.

[default machine]

The `[default machine]` keyword defines a section containing parameters that control the operation of the ASP Server on each computer. Parameters are listed in the following table.

Parameter	Explanation
serverroot	The install root of the ASP Server. **Warning:** DO NOT change this parameter.
serverlib	ASP Server libraries. **Warning:** DO NOT change this parameter.
caspd_pid (optional)	The name and location of the process ID (PID) file for the Sun ONE ASP daemon.
license	The absolute path name of the directory containing the Sun ONE ASP license file.
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, especially if you are running Sun ONE Web Server. When inherit_user=1 (**Inherit User Security** mode), 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 case even if a different user and group is specified in the [default machine] section of casp.cnfg. This mode is available only for Sun ONE ASP running with the Apache Web server (for Sun ONE Web server, see below). When inherit_user=0 (**Defined User Security** mode), the ASP Server runs with the permissions of the user who started the ASP Server, unless a different user and group is specified in casp.cnfg. This mode is available for both the Apache and Sun ONE Web servers. **Warning:** When inherit_user=0 and no user and 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. Do not set inherit_user=0 unless you also define a user and group for the ASP Server to run under. For more information about the security modes and their implications, see "Setting the Security Mode" on page 57.

Parameter	Explanation
user (optional)	The username for the account under which the ASP Server runs. Make sure that this user has permission to open Sun ONE 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 who started the ASP Server. **Note:** To start the ASP Server under a non-root user account, do the following: - chmod otw [C-ASP_INSTALL_DIR]/asp-server-#### - chmod otw [C-ASP_INSTALL_DIR]/asp-server-####/casp.cnfg - chmod otr [C-ASP_INSTALL_DIR]/asp-server-####/odbc.ini These steps are required for ADO functionality.
group (optional)	The group name for the account under which the ASP Server runs. Make sure that this group has permission to open Sun ONE ASP configuration files such as casp.cnfg and odbc.ini. The user starting the ASP Server by 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 who started the ASP Server. See the note pertaining to non-root user accounts under the **user** parameter (above).
asplogfile (optional)	The full file path for where ASP errors will be logged.
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.
javasupport (yes/no)	**Yes** (the default) enables Java support, which is required for the Sun ONE ASP Chili!Beans wrapper, and for XML and ASP servlet interface functionality. **No** disables Java support. Because Java support can affect server performance, it is a good idea to enable it only when using Chili!Beans.
codepage	The value for the code page associated with the specified locale (LCID).
lcid	The value for the specified geographic locale (LCID).

Parameter	Explanation
javasecurity manager (yes/no)	**Yes**, the default, enables the Java virtual machine Security Manager. **No** disables it. For security reasons, the Java virtual machine Security Manager should be enabled in multi-user environments in which users supply their own Java classes.
JavaUserClsPath (optional)	Specifies additional paths to look for Java classes. Full directory paths can be separated by semicolons (;).

[virtual hosts]

(Optional) The `[virtual hosts]` keyword defines a section in which to configure the ASP Server to work with virtual hosts or virtual servers. For more information, see "Defining Applications on UNIX" on page 525, "Enabling ASP for a Virtual Host" on page 54, and "Defining Applications in a Shared Environment" on page 74.

Parameter	Explanation
allow_all (yes/no) (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 virtual hosts defined in the Web server configuration file.
hostID(s) (optional)	This setting is a line-delimited list of hostnames that identify which virtual hosts or virtual servers are allowed to handle requests for ASP pages. 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.

[default application]

The `[default application]` keyword defines a section containing parameters that control the behavior of ASP applications. Parameters are listed in the following table.

Note

If you make any changes to casp.cnfg, you must restart the Sun ONE ASP Server. If you make any changes to the [default application] section in casp.cnfg, you must restart both the Sun ONE ASP Server and the Web server.

Parameter	Explanation
bufferingon (yes/no)	**Yes** (the default) enables script buffering.
sessiontimeout	Amount of time in seconds that the ASP Server waits for a new page request before canceling the session.

Parameter	Explanation
scripttimeout	Amount of time in seconds the ASP Server waits for an ASP page to finish processing before canceling the request. A value for ScriptTimeout specified in a script will always override the value set in casp.cnfg. **Note:** If deadlocktimeout (below) is set to a value lower than scripttimeout, the ASP engine will restart once the time specified for deadlocktimeout has elapsed.
deadlocktimeout	Amount of time, in seconds, that should elapse before the ASP Server is considered deadlocked and the engine is restarted. **Note:** If deadlocktimeout is set to a value lower than scripttimeout, the ASP engine will restart once the time specified for deadlocktimeout has elapsed.
allowsessionstate (yes/no)	**Yes** (the default) enables the use of the **Session** object in ASP scripts.
enableparentpaths (yes/no)	**No**, the default, limits file system access by the **FileSystemObject** object to the application directory and subdirectories, and disables the use of `" . . / "` syntax. **Yes** enables access to the file system by the **FileSystemObject** object outside the ASP application directory and the use of `" . . / "` syntax in **#INCLUDE** and **Server.MapPath** statements. **Caution:** Changing **Enable parent paths** to **yes** can affect the security of your server. Before you change this setting, make sure that the 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. **Note:** The **Enable parent paths** setting does not add any restrictions to executing Java code. For example, if you want to restrict Java code to access files within the application directory, the proper permissions should be in the bean.policy file.
defaultlanguage (vbscript or javascript)	Specifies the default script interpreter. This value can either be **vbscript** or **javascript**. (For more information about scripting engines provided with Sun ONE ASP, see "Chapter 15, Scripting Languages Reference" on page 503.)
showdefaulterror (yes/no)	When set to **Yes**, the value of **defaulterror** (below) is displayed when the ASP Server encounters a run-time error.
defaulterror	The error message displayed when **showdefaulterror** (above) is set to **Yes**.
500errordocument	When a script-parsing or run-time error is encountered, the server redirects to this custom error page. The default [C-ASP_INSTALL_DIR]/admin/sys/500-100.asp page displays the error information (see "Remarks: ASP Server Object GetLastError Method" on page 256).

Parameter	Explanation
500-15errordocument	When the browser requests the global.asa file and an error is encountered, the server redirects to this custom error page. The default [C-ASP_INSTALL_DIR]/admin/sys/500-15.htm page displays the error information (see "Remarks: ASP Server Object GetLastError Method" on page 256).

[ADO]

The `[ADO]` keyword defines a section containing parameters that control the operation of the ADO component. Parameters are listed in the following table.

Parameter	Explanation
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. This will not be functional if inherit_user is set or if the caspeng process doesn't have permission to access the file.
connectiontimeout	Amount of time to wait for establishing a connection before terminating the attempt and generating an error.
maxlongfieldlength	Maximum long field length in bytes. By default this value is 65535. If the data you pass to a database exceeds this limit, ADO will throw an error. You can increase this value as needed.

[admin]

The `[admin]` keyword defines a section that controls the operation of the Sun ONE ASP Administration Console. Parameters are listed in the following table.

Parameter	Explanation
friendlyname	Name displayed in the Administration Console **Server Monitoring** and **Live Monitoring** pages.
restart (yes/no)	Used by the Administration Console to flag that the ASP Server needs to be restarted. **Warning:** Use the Administration Console to change this setting. DO NOT change this setting manually.

[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" on page 47.

Parameter	Explanation
use_aliases (yes/no)	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.
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 file name 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.
/casp-sys	Location of Administration Console support files and default error page.
/dbms	Sun ONE ASP DBMS for MySQL application.
/caspdoc	Absolute path name of the directory containing the Sun ONE ASP product documentation.
/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 document root directory of the Web server as the location of the "default" ASP application.

[Components Security]

The `[Components Security]` keyword defines a section that allows you to enable components such as the SMTP, POP3, and Upload components. Parameters are listed in the following table.

Parameter	Explanation
CDONTS.NewMail	**On** enables the Sun ONE ASP SMTP component.
Chili.POP3	**On** enables the Sun ONE ASP POP3 component.
ChiliUpload	**On** enables the Sun ONE ASP Upload component.
MaxTransferSize	Specifies the maximum number of bytes that can be uploaded by the Upload component.

[Product Update]

The `[Product Update]` keyword defines a section that controls when the Administration Console should automatically check for product updates. Parameters are listed in the following table.

Parameter	Explanation
Date	The Administration Console will automatically check for updates when you go to the **Server Management** page 90 days after this date.

Defining Applications on UNIX

This section describes the options that are available for defining Sun ONE Active Server Pages applications on UNIX-based systems.

Caution

Take great care when making the changes described in this section. Changes you make could require a complete reinstall of Sun ONE Active Server Pages and could void your eligibility for customer support. Back up your data before making any changes.

Most of the configuration settings described in this section are easily accessed from the Sun ONE ASP Administration Console. It is strongly recommended that you use the Administration Console whenever possible, as described in "Chapter 2, Using the Administration Console" on page 17.

With Sun ONE ASP running on a UNIX system with any supported Web server, you can define an ASP application by using the following methods:

- Using the Sun ONE ASP Administration Console. For more information, see "Enabling ASP for a Virtual Host" on page 54.
- Adding an entry to the `[applications]` section of the Sun ONE ASP configuration file, casp.cnfg. For more information, see "Editing the Sun ONE ASP Configuration File" on page 517.
- 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 ONE 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 ONE 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 ONE 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 ONE ASP casp.cnfg file, or by adding an alias to your Web server configuration. For more information, see "Editing the Sun ONE ASP Configuration File" on page 517.

Defining an Application on Sun ONE 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. Document roots are also ASP applications. 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" in the **Content Mgmt** tab of the **Class Manager**.

If you have configured support for virtual servers, you can define ASP applications 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 ONE 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.

Note

This last method does not require a restart of the ASP Server and the Web server

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. Document roots are also ASP applications. You can use the **Alias** parameter in httpd.conf 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 ONE Active Server Pages is installed. An ASP application can also be defined by adding an entry to the `[applications]` section of casp.cnfg. This applies to the "real host" only.

If you have configured support for virtual hosts, you can define ASP applications on Apache Web Server as follows:

- 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 ONE 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.

Note

This method does not require a restart of both the ASP Server and the Web server. The other methods listed here do require restarts.

- 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 ONE Active Server Pages 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 ONE 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 ONE ASP and could void your eligibility for customer support. Back up your data before making any changes.

Most of the configuration settings described in this section are easily accessed from the Sun ONE ASP Administration Console. It is strongly recommended that you use the Administration Console whenever possible, as described in "Chapter 2, Using the Administration Console" on page 17.

Relocating the Registry File

One of the key Sun ONE Active Server Pages 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 ONE ASP installation, the installer creates a script named chsetup.sh in the Sun ONE 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 in which the file system supports file locking. For the following steps, this new directory is 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 ONE Active Server Pages PID Files

For users who want to install Sun ONE Active Server Pages to a shared file system, but move writeable Sun ONE ASP files to a local file system, Sun ONE ASP provides a mechanism to allow for this. For single machine Sun ONE 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 ONE 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.

Configuring a Non-DSO Apache Web Server

This procedure applies if you have installed Sun ONE Active Server Pages on a computer running an Apache Web server that does not have Dynamic Shared Object (DSO) support.

Note

As described in "Supported in This Release" on page 5, non-DSO Apache Web servers have not been certified to run with Sun ONE Active Server Pages, and their use is not supported by Customer Support.

When you install Sun ONE ASP on a computer running an Apache Web server, the setup program automatically links Sun ONE ASP to Apache through Apache's module facility. The setup program uses pre-built Sun ONE Active Server Pages 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 ONE ASP to the Apache Web server.

Note

The Sun ONE 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 Configuration Files" on page 81.

To manually link the Sun ONE ASP module to an Apache Web server that does not have DSO support, install Sun ONE 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 ONE 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 ONE 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. 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 ONE ASP Server, as described in "Stopping and Restarting the ASP Server (Admin Console)" on page 41.

Starting the Apache Web Server in SSL Mode

The Sun ONE Active Server Pages Administration Console can be used to start, stop, and restart the Web server with which the Sun ONE 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 ONE 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
  ```

Glossaries

This chapter includes two glossaries: A glossary with general terms you might encounter when administering Sun ONE ASP and developing ASP applications, and a glossary with terms specific to the Sun ONE ASP Administration Console.

In this chapter:

General Glossary

This glossary defines general terms you might encounter when administering Sun ONE ASP and developing ASP applications.

A

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\file name.suffix

UNIX-based systems use this format:

/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 ASP (Active Server Pages), but do not require a Windows interface.

ActiveX

ActiveX is a set of technologies built on the COM (Component Object Model) that enable software components, regardless of the language in which they were

developed, to work together in a networked environment. Although ActiveX technologies are used primarily to develop interactive Web content, they also can 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 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 ONE Active Server Pages includes script interpreters for both VBScript and JScript. These interpreters are Sun ONE ASP VBScript and Sun ONE ASP JavaScript, which provide functionality equivalent to version 5.5 of Microsoft VBScript and JScript.

Administration Console

The Sun ONE ASP Administration Console is a browser-based application used for managing Sun ONE ASP. It enables administrators to configure and control the Sun ONE ASP Server and its bindings to Web servers and database servers from a Web browser, either locally or remotely. Most configuration settings are accessible from the Administration Console. Whenever possible, you should use the Administration Console for product configuration.

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 ONE ASP.

See "Chapter 2, Using the Administration Console" on page 17.

ADO (ActiveX Data Objects)

ADO 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 ONE Active Server Pages 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. The implementation of ADO used with Sun ONE ASP is called ADODB.

See "Chapter 11, ADO Component Reference" on page 301.

Apache Web Server

Apache Web Server is a Web server that is developed and maintained through an open-source project. For the list of Apache Web Server versions supported in Sun ONE ASP, see "Supported in This Release" on page 5.

Application object

The **Application** object is one of the built-in objects included in Sun ONE Active Server Pages. 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 "ASP Application Object" on page 216.

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 ONE ASP Server is an application server that enables you to use ASP (Active Server Pages) specification to execute database queries, execute business logic, and generate the presentation layer for Web applications.

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.

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 ONE Active Server Pages 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 ONE ASP Server, which processes the scripts, generates an HTML page, and sends it back to the browser.

ASPError object

The **ASPError** object is one of the built-in objects included in Sun ONE Active Server Pages. The **ASPError** object reports error information, and can be used to obtain information about an error condition that has occurred in script in an ASP page.

See "ASPError Object" on page 222.

ASP application

An ASP application is a set of Active Server Pages files contained within a single root directory. For the Sun ONE 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.

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.

ASP engine

The Sun ONE ASP Server uses an ASP engine to execute ASP scripts. A single ASP engine executes as many threads as are specified in the Sun ONE ASP Administration Console.

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 file name 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 ONE Active Server Pages, you can write scripts in VBScript or JScript. Any valid HTML page can be a valid ASP page, enabling Web developers to easily transform a static Web site into a dynamic one by adding ASP scripts to existing documents. Saving the page with an .asp file name extension tells the Web server how to process the script commands.

See "Chapter 8, Building Sun ONE ASP Applications" on page 181.

ASP script

An ASP script is a server-side script that is included on an ASP (Active Server Pages) page. With Sun ONE Active Server Pages, scripts can be written in either VBScript or JScript.

ASP Server

When a browser requests an ASP page, the Web server passes execution to the Sun ONE 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.

ASP session

An ASP session is created by using the Sun ONE Active Server Pages 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.

B

Built-in objects

Sun ONE Active Server Pages includes built-in or intrinsic objects that handle many common programming tasks. The objects are **Application**, **ASPError**, **Request**, **Response**, **Server**, and **Session**. 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 low-level programming. Built-in objects are included on all ASP pages, and do not need to be created before they can be used.

See "Chapter 9, ASP Built-in Objects Reference" on page 215.

C

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 (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.

CASP

CASP is a commonly used abbreviation for Sun ONE Active Server Pages, formerly known as Chili!Soft ASP (thus the "CASP" abbreviation). CASP is used for the default installation directory name, and for several virtual directories installed by the Sun ONE ASP setup program.

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 technology is a high-performance alternative to CGI.

Chili!Beans

Sun ONE ASP Chili!Beans enables access to Java classes (including JavaBeans and Enterprise JavaBeans) from the ASP environment by wrapping Java classes in a COM layer. This enables Java objects to be used by COM controllers, such as ActiveX scripting engines like VBScript. The Chili!Beans ActiveX control is included in Sun ONE Active Server Pages, and is designed to work with Java virtual machine (JVM) versions 1.4 or greater.

See "Chapter 12, Chili!Beans Component Reference" on page 465.

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 uses the Internet's global DNS (Domain Naming Service) for scalability. It 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, JScript, and ECMAScript. Unlike server-side scripting, client-side scripting requires no special implementation on the Web server, but it does require the appropriate support on the client browser. Sun ONE Active Server Pages applications involve server-side scripts and can also include client-side scripts.

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 use the Sun ONE Active Server Pages Administration Console to set the correct code page and LCID for a given language.

See "Configuring International Support" on page 43.

COM (Component Object Model)

COM 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. Sun ONE Active Server Pages supports COM components on Windows. On UNIX and Linux, components based on Java™ technology are required (JavaBeans™).

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.

See "Using Custom Server Components" on page 196 and "Chapter 12, Chili!Beans Component Reference" on page 465.

Connection pooling

To improve server performance, you can configure the Sun ONE 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.

Connection string

A connection string defines the source of data for an external database. On a Sun ONE ASP page, a connection string must 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.

Cookie

A cookie is a file of encoded information, stored on a user's computer, which identifies the user's computer during current and subsequent visits to a Web site.

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. CORBA objects can be accessed by using Sun ONE ASP's COM-to-Java bridge.

D

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 DSN (data source name) and logon information. For example, a data connection for a MySQL 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 "Chapter 6, Configuring a Database" on page 103.

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 ADO and the ODBC interface, the JDBC interface, or a CORBA interface broker. Sun ONE Active Server Pages includes an ODBC driver for connecting to DB2.

dBASE

dBASE is a database management system. Sun ONE Active Server Pages 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 to 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 ONE ASP supports generate code that is compatible with Sun ONE ASP. These tools include Adobe GoLive, Macromedia UltraDev, and others.

DLL (dynamic-link library)

A DLL is a code file containing functions that can be called from other executable code (either an application or another DLL). Programmers use DLLs to 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 ONE ASP engine and all components run on a Windows emulation layer. The equivalent of a DLL on these emulation layers are files having an *.so (Shared Object) file name extension.

DOM (Document Object Model)

DOM is a programming interface specification of the W3C (World Wide Web Consortium) that enables programmers to create and modify HTML pages and XML documents as full-fledged program objects. Currently, HTML and XML 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 (data source name)

DSN refers to a collection of information required to connect an ASP application to a particular ODBC-compliant database. The ODBC Manager uses this information to create the database connection. Sun ONE Active Server Pages supports two types of DSNs: system DSNs and file DSNs.

See "Configuring Data Source Names (DSNs)" on page 105 and "Connecting to a Database" on page 197.

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 for them to work in the new environment.

See "Creating Connection Strings" on page 197.

DSO (Dynamic Shared Object)

Sun ONE Active Server Pages communicates with Apache Web Server through an object module. In the DSO version, Apache Web Server object module entries are loaded on an as-needed basis. Sun ONE ASP 4.0 does not support non-DSO versions of Apache Web Server.

E

ECMAScript (European Computer Manufacturers Association Script)

ECMAScript 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.

EJB (Enterprise JavaBeans)

Enterprise JavaBeans™ 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 ONE Active Server Pages enables ASP applications to access EJB through Sun ONE ASP's COM-to-Java bridge.

EJB is built on the JavaBeans technology for distributing program components (called beans) to clients in a network. With EJB, enterprises can control changes at the server, instead of having 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, 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 CGI (Common Gateway Interface) 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 one that, unlike a session bean, has persistence and can retain its original behavior or state.

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.

F

File DSN (data source name)

File DSN 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 file name extension. Developers can incorporate the database information into a connection string by referring to the file DSN rather than having to specify the values for each required parameter. A file DSN can be shared among several users. Sun ONE ASP also supports system DSNs.

See "Configuring Data Source Names (DSNs)" on page 105 and "Connecting to a Database" on page 197.

FileSystemObject object

In ASP applications, the **FileSystemObject** object provides access to a computer's file system. The object can perform simple functions such as opening and closing files. Various methods can be applied to the **FileSystemObject** object to affect the organization and properties of a file system.

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 Server Extensions

Sun ONE Active Server Pages supports but does not install Microsoft FrontPage Server Extensions. FrontPage Server Extensions are a set of server-side applications (i.e., CGI programs) that enable you 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 IIS.

Sun ONE 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.

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 because each subroutine or function can be tested separately.

G

Global.asa

Global.asa is a file that contains information that is global to a specific ASP application. The 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 developers to specify event procedures and declare objects that have session or application scope.

See "Using the Global.asa File" on page 189.

H

Hostname

A hostname is the valid DNS (Domain Naming Service) name for a Web server.

HTML (Hypertext Markup Language)

HTML 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 W3C (World Wide Web Consortium) and is generally adhered to by the major browsers (although both Internet Explorer and Netscape do implement some features differently and provide nonstandard extensions).

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 FTP (File Transfer Protocol) and Gopher. The Web is a network consisting of these types of servers. Among the most popular HTTP servers are the Sun ONE Web Server (formerly iPlanet Web Server, Enterprise Edition), and the Apache Web Server.

I

IIS (Internet Information Server)

IIS is Microsoft's Web server that runs on the Windows NT and Windows 2000 platforms only.

iPlanet Web Server, Enterprise Edition (Sun ONE Web Server)

iPlanet™ Web Server, Enterprise Edition is Web server software originally developed by Netscape and now offered by Sun Microsystems. iPlanet Web Server, Enterprise Edition is now called Sun ONE Web Server.

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 CGI (Common Gateway Interface), but offers performance improvements on Windows-based Web servers.

J

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 have Java interpreters and run-time environments known as Java virtual machines.

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.

JavaBeans

JavaBeans™ is an object-oriented programming interface developed by Sun Microsystems that enables developers to build reusable 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 ONE Active Server Pages enables ASP applications to access Java objects and classes through Sun ONE ASP 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 JavaBeans Development Kit 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 provide.

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.

JDBC (Java Database Connectivity)

JDBC is a data access interface based on ODBC (Open Database Connectivity) 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.

JVM (Java virtual machine)

The Java™ virtual machine is the cornerstone of the Sun Microsystem's 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.

K

Key

In application programming, a key is the code for deciphering encrypted data.

L

LCID (Local Language Identifier)

An 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 use the Sun ONE ASP Administration Console to set the LCID.

See "Configuring International Support" on page 43.

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.

M

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.

MDAC (Microsoft Data Access Components)

Microsoft Data Access Components are comprised of ADO and RDS (Remote Data Service), ODBC, and the Microsoft OLE DB Provider for ODBC, which are released, documented, and supported together. Of these, Sun ONE Active Server Pages supports ADO and ODBC only.

Moniker

A moniker is a name that uniquely identifies a 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 (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 ONE Active Server Pages 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.

MySQL

MySQL is an open source database and relational database management system. MySQL uses an implementation of SQL (Structured Query Language). Sun ONE Active Server Pages includes an ODBC driver for connecting to MySQL.

Sun ONE ASP also includes two database tools related to MySQL: Sun ONE ASP Database Publisher (Database Publisher), and Sun ONE ASP Database Management System for MySQL (DBMS). Database Publisher is a client/server application that enables a Microsoft Access database running on Windows to be published to a MySQL database running on UNIX or Linux. DBMS is a database administration system for MySQL, enabling MySQL databases to be administered from a user-friendly administration console instead of strictly from the command line.

For more information about these tools, see "Chapter 7, Using Database Tools" on page 135.

N

NSAPI

NSAPI is the API for Sun ONE Web servers. NSAPI enables programmers to create Web-based applications that are more sophisticated and run much faster than applications based on CGI.

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, in which 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.

O

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++.

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).

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 ONE Active Server Pages include ODBC drivers for a number of different databases. ODBC drivers enable you to connect to the databases and access their data.

See "Viewing the List of ODBC Drivers" on page 104 and "Configuring Database Parameters" on page 115.

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 ONE ASP includes ODBC drivers for a number of different databases, and enables you to specify drivers from the Sun ONE ASP Administration Console.

ODBC Manager

The ODBC Manager manages connections between ODBC drivers and databases by using information stored in the DSN (data source name).

P

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

UNIX-based systems use this format:

/directory/subdirectory/filename

In UNIX, the storage drive location is not an explicit part of the path name.

POP3 (Post Office Protocol)

POP3 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 Sun ONE ASP's SpicePack with the Chili!Mail component.

Port

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 HTTP, have port numbers that are preassigned by the IANA (Internet Assigned Numbers Authority). 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 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 APIs (application programming interfaces) 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, 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 ONE Active Server Pages supports Java classes through Sun ONE ASP Chili!Beans.

For more information about Chili!Beans, see "Chapter 12, Chili!Beans Component Reference" on page 465.

PostgreSQL

PostgreSQL is a sophisticated Object-Relational Database Management System, supporting almost all SQL constructs including subselects, transactions, and user-defined types and functions. Sun ONE Active Server Pages includes an ODBC driver for connecting to PostgreSQL.

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.

R

Request object

The **Request** object is a Sun ONE Active Server Pages built-in object that retrieves the values the client browser passed to the server during an HTTP request.

See "ASP Request Object" on page 224.

Response object

The **Response** object is a Sun ONE Active Server Pages built-in object that can be used to control output sent to the client.

See "ASP Response Object" on page 235.

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.

S

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, VBScript, and JScript. Sun ONE Active Server Pages supports VBScript and JScript.

SequeLink

SequeLink is server-based middleware provided by DataDirect Technologies. Sun ONE Active Server Pages includes the SequeLink client for connecting to remote Microsoft Access and Microsoft SQL Server databases running on Windows systems.

See "Configuring SequeLink" on page 128.

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 ONE Active Server Pages, when a browser requests an ASP page, the Web server sends the request to the Sun ONE ASP Server. The Sun ONE 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. Server-side scripting enables users to receive customized pages based on browser capabilities, user preferences, and content from a server-side database.

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 ONE Active Server Pages built-in object that provides access to methods and properties on the server. Most of its methods and properties serve as utility functions.

See "ASP Server Object" on page 251.

Session object

The **Session** object is a Sun ONE Active Server Pages 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, but rather persist for the entire user session. The Sun ONE 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 developers 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 "ASP Session Object" on page 261.

Session state

Session state refers to information that the Sun ONE 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.

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 or virtual servers. 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.

SMTP (Simple Mail Transfer Protocol)

SMTP 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 port 25. You can find the details of SMTP in Request for Comments 821 of the IETF (Internet Engineering Task Force).

Solaris

The Solaris™ Operating Environment is the platform provided by Sun Microsystems for its family of Scalable Processor Architecture-based processors and for Intel-based processors. Sun emphasizes the system's availability, 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 JDK™ (Java Development Kit).

SpicePack

The SpicePack is a collection of COM components that handle commonly used ASP application functionality. The SpicePack is no longer a separate product, but is now bundled with Sun ONE Active Server Pages. The components are Chili!Mail, Chili!POP3, and Chili!Upload. These components can be instantiated and called from ASP pages to send and receive e-mail and upload files from client browsers.

See "Chapter 14, SpicePack Component Reference" on page 483.

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.

SQL Server

Microsoft SQL Server is SQL server software offered by Microsoft. Sun ONE Active Server Pages is fully compatible with Microsoft SQL Server systems.

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 IETF (Internet Engineering Task Force) has generated a successor of SSL, a network standard called TLS (Transport Layer Security).

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 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 DSN (data source name)

A system 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 ONE Active Server Pages 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 "Configuring Data Source Names (DSNs)" on page 105 and "Connecting to a Database" on page 197.

T

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.

V

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.

VBScript (Visual Basic Scripting Edition)

VBScript is a scripting language developed by Microsoft that is based on the more complex VB (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 VB programmer, VBScript is the easiest scripting language to learn. (JScript is easier for C/C++ programmers.) Sun ONE Active Server Pages supports both VBScript and JScript.

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 ONE Active Server Pages

supports virtual hosts (referred to as virtual servers on Sun ONE Web Server).

W

W3C (World Wide Web Consortium)

The 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.

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 FTP (File Transfer Protocol) and Gopher. The Web is a network consisting of these types of servers. HTTP servers are commonly referred to as Web servers. Among the most popular Web servers are the Sun ONE Web Server (formerly iPlanet Web Server, Enterprise Edition), and the Apache Web Server.

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 ONE Active Server Pages, developers can use the built-in **Session** object to maintain session information for each user, providing consistent user sessions on the Web.

X

XML (Extensible Markup Language)

XML is a simplified subset of SGML (Standard Generalized Markup Language) 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 APIs (application programming interfaces) 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 W3C DOM (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 is a language that defines the rules for mapping structured XML data and documents. Derived from DSSSL (Document Style Semantics and Specification Language), XSL also has roots in the SGML (Standard Generalized Markup Language) 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 CSS (Cascading Style Sheets), but rather is designed to handle the new capabilities of XML that CSS cannot.

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.

Z

Zeus Web Server

The Zeus Web Server is a Web server produced by Zeus Technologies. The Zeus Web Server is not supported in Sun ONE Active Server Pages 4.0.

Administration Console Glossary

This glossary defines elements that appear in the Sun ONE ASP Administration Console graphical user interface (GUI).

A

C

D

M

N

O

P

R

S

T

U

V

Active virtual hosts

This field displays the number of virtual hosts enabled for this ASP Server.

See also:

"Enabling ASP for a Virtual Host" on page 54
"Monitoring ASP Server Performance" on page 61

ActiveX Data Object Connection Setting

ActiveX Data Objects (ADO) provides a set of objects used for connecting to databases that conform to the ODBC standard. ADO connection settings are configured on this page.

See also:

"Configuring ADO Connections" on page 131

Add new DSN

Parameters for new DSNs (data source names) are configured by clicking **Add new DSN**, which displays the **New Data Source Name** page.

A DSN refers to a collection of information used to connect an ASP application to a particular ODBC-compliant database. The ODBC Manager uses this information to create the database connection.

See also:

"Configuring Data Source Names (DSNs)" on page 105

"Creating Database Connections (ASP Server)" on page 44

Add user

The Sun ONE ASP Administration Console is used to add, edit, and delete usernames and passwords. To add a user, specify the username, password, and password confirmation in the corresponding boxes on the **Users** screen, and then click **Add user**.

See also:

"Configuring Usernames and Passwords" on page 21

Allow session state

Yes (the default) enables the **Session** object so it can be used in ASP applications.

See also:

"Enabling Session State" on page 42

"Changing ASP Server Settings" on page 37

Application name

Specify the name of the virtual directory to create and enable as an ASP application. This virtual directory refers to a physical directory in the file system (specified in the **Directory** box) that contains the ASP application files.

See also:

"Configuring ASP Applications" on page 47

"Defining Applications in a Shared Environment" on page 74

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.

"Configuring Database Parameters" on page 115

"Chapter 6, Configuring a Database" on page 103

ASP Applications

The Administration Console **Applications** page displays the list of ASP applications that are currently defined on the ASP Server, and provides access to settings for adding, configuring, and removing ASP applications.

An ASP application is a set of ASP files contained within a single root directory. For the ASP Server to recognize an ASP application, the root directory must be defined as an application on the ASP Server, and 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:

"Defining ASP Applications (ASP Server)" on page 46

ASP errors logging file

To create a log file and enable ASP error logging, specify the name of the log file. You cannot use the name of a file that already exists in the directory. If this box is empty, no logging is performed.

When you specify the name of the log file, the ASP Server creates the file in the following directory and begins logging to it:

/[C-ASP_INSTALL_DIR]/logs

Where [C-ASP_INSTALL_DIR] is the path name of the Sun ONE ASP installation directory (/opt/casp by default).

See also:

"Enabling ASP Errors Logging" on page 63

"Changing ASP Server Settings" on page 37

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. This option is disabled by default, which returns **SQL_NULL_DATA** for

the result columns **COLUMN_DEF** and **REMARKS**. Enabling this option reduces the performance of your queries.

See also:

"Configuring Database Parameters" on page 115

"Chapter 6, Configuring a Database" on page 103

change password

The Sun ONE ASP Administration Console is used to add, edit, and delete usernames and passwords. To change the password for an existing user, you must be logged on as that user (that is, any administrator can change his or her own password, but not that of any other administrator). On the **Users** page, click **change password**. In the **Change Password** dialog that displays, specify and confirm the new password, and then click **OK**.

See also:

"Configuring Usernames and Passwords" on page 21

Chili!Beans

The Sun ONE ASP Chili!Beans ActiveX control is a wrapper that enables Java objects to be used by COM controllers, such as ActiveX scripting engines like VBScript. The control is designed to work with Java virtual machines (JVM) versions 1.4 or greater. To use Chili!Beans, a Java runtime environment must be installed, and Chili!Beans must be enabled from the Sun ONE ASP Administration Console.

See also:

"Chili!Beans Component Reference" on page 465

Chili!Mail

Chili!Mail is a Sun ONE ASP SpicePack component that enables users to send e-mail messages from an ASP page to an SMTP server. The Chili!Mail component is compatible with the **NewMail** object included with the Microsoft Internet Information Services (IIS) CDONTS component. However, there are differences regarding certain properties and methods of the **NewMail** object.

See also:

"Chili!Mail (SMTP)" on page 484

Chili!POP3

Chili!POP3 is a Sun ONE ASP SpicePack component that 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, and the message

interface exposes all properties of a single message. Additional interfaces are exposed to support retrieval of message lists and message attachments.

See also:

"Chili!POP3 (POP3)" on page 491

Chili!Upload

Chili!Upload is a Sun ONE ASP SpicePack component that enables users to save files uploaded by site visitors to the server. Chili!Upload supports the simultaneous upload of multiple files. Form data can also be processed while files are uploading.

See also:

"Chili!Upload (File Upload)" on page 499

Collection

Specify this attribute only if the DB2 database is running on OS/390.

This is the name that identifies a group of packages. These packages include the Connect ODBC for DB2 Wire Protocol driver packages. The default is DATADIRECT00.

See also:

"Configuring Database Parameters" on page 115

"Chapter 6, Configuring a Database" on page 103

Components

The Administration Console **Components** page provides access to settings for the Sun ONE ASP SpicePack and Chili!Beans components. The components are enabled or disabled on this page.

The SpicePack is a set of 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.

See also:

"SpicePack Component Reference" on page 483

"Chili!Beans Component Reference" on page 465

Confirm Password

Retype the password to confirm it.

Connection pool size

To improve server performance, the ASP Server can be configured to share 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.

There is no maximum number of connections that can be pooled. Setting this number to 0 (zero) disables connection pooling. The default is 25.

See also:

"Configuring ADO Connections" on page 131

"Pooling Database Connections" on page 72

Create database

Select or clear the **Create database** box to specify global Create privileges.

If this box is selected, client-side users with appropriate privileges can create a new database on the MySQL server to which to publish an Access database.

If this box is cleared, all users are barred from creating a new database, regardless of user privileges. If users are not allowed to create a new database, a database must be supplied on the MySQL server for which users have all permissions.

See also:

"Administering Database Publisher" on page 136

"Database Publisher" on page 135

Current number of sessions

This field displays the number of sessions currently being handled by the ASP Server.

See also:

"Monitoring ASP Server Performance" on page 61

Data Source Names

Parameters for system DSNs (data source names) are configured on the Administration Console **Data Source Names** page. A system DSN is a collection of information stored on the ASP Server that is used by the ODBC Manager for connecting an ASP application to a particular ODBC-compliant database.

See also:

"Configuring Data Source Names (DSNs)" on page 105

"Configuring Database Parameters" on page 115

"Creating Database Connections (ASP Server)" on page 44

"Chapter 6, Configuring a Database" on page 103

"Connecting to a Database" on page 197

Database

This field requires different information for different databases:

- For DB2 databases, specify the DSN name of the database. This must be the same as the catalogued name for the database.
- For dBASE databases, specify the absolute path name of the directory in which the database files reside.
- For Informix, Microsoft SQL Server, MySQL, PostgreSQL, and Sybase databases, specify the name of the database.
- For Microsoft Access databases, specify the absolute path name of the Access MDB file on the Windows server.
- For the Text driver, specify the directory in which the text files are stored. If left empty, the current working directory is used.

Ask your database administrator for this information.

See also:

"Configuring Database Parameters" on page 115

"Chapter 6, Configuring a Database" on page 103

Databases

The Administration Console **Databases** page is used to configure DSNs, ODBC drivers, ADO settings, and database environment variables.

Sun ONE ASP is compatible with and provides ODBC drivers for a number of databases. Supported databases include DB2, dBASE, Informix, Microsoft Access, Microsoft SQL Server, MySQL, Oracle, PostgreSQL, and Sybase.

See also:

"Supported in This Release" on page 5

"Viewing the List of ODBC Drivers" on page 104

"Creating Database Connections (ASP Server)" on page 44

"Chapter 6, Configuring a Database" on page 103

Database type

From the drop-down list on the **New Data Source Name** page, select the type of database to configure for this DSN. On the **Edit Data Source Name** page, **Database type** simply lists the type of database configured for this DSN.

Sun ONE ASP supports the following types of databases, and includes their ODBC drivers:

- DB2 Universal Database (UDB) 7.1
- dBASE 5
- Informix Dynamic Server 9.x
- Informix Dynamic Server 2000 (9.20)
- Microsoft Access 2000, 97, and 95 (via SequeLink 5.3)
- Microsoft SQL Server 7.0 and 2000 (SP1)
- Microsoft SQL Server 6.5 (via SequeLink 5.3)
- MySQL 3.23
- Oracle 8i (8.1.7) and 9i
- PostgreSQL 7.1.3
- Sybase Adaptive Server Enterprise 11.9.2 and 12.5
- Text files

Note

Sun ONE ASP does not support all databases on all platforms. To see the list of installed drivers for your platform, see "Supported in This Release" on page 5.

See also:

"Chapter 6, Configuring a Database" on page 103

"Configuring Database Parameters" on page 115

Deadlock timeout

This specifies the amount of time, in seconds, that should elapse before the ASP Server is considered deadlocked and the engine is restarted. The default is 600 seconds (10 minutes).

See also:

"Configuring Engine Deadlock Recovery" on page 69

Description

Specify a description of the DSN to help distinguish it from others.

See also:

"Configuring Data Source Names (DSNs)" on page 105

Directory

Specify the absolute path name of the directory containing the ASP application files. The virtual directory specified in the **Application Name** box is associated with this directory.

See also:

"Configuring ASP Applications" on page 47

"Defining ASP Applications (ASP Server)" on page 46

"Defining Applications in a Shared Environment" on page 74

Driver

This is the installed ODBC driver specified for this DSN. Sun ONE ASP includes ODBC drivers for a number of databases. An ODBC driver is a module that enables a database to be accessed through ODBC (Open Database Connectivity). A separate driver is required for each type of database.

See also:

"Supported in This Release" on page 5

"Viewing the List of ODBC Drivers" on page 104

"Configuring Data Source Names (DSNs)" on page 105

"Configuring Database Parameters" on page 115

DSN or Data Source Name

Specify the name of this DSN (data source name).

A DSN refers to a collection of information used to connect an ASP application to a particular ODBC-compliant database. The ODBC Manager uses this information to create the database connection.

See also:

"Configuring Data Source Names (DSNs)" on page 105

Enable

When the **Enable** box is selected, the Sun ONE ASP Database Publisher and DBMS applications are enabled for client-side use. If this box is not selected, users will not be able to use the applications.

See also:

"Administering DBMS" on page 149

"Administering Database Publisher" on page 136

EnableDescribeParam

When this option is enabled (the check box is selected), all StoredProcedure arguments are returned as string types. This option is enabled by default.

See also:

"Configuring Database Parameters" on page 115

"Chapter 6, Configuring a Database" on page 103

EnableStaticCursorsForLongData

When this option is enabled (the check box is selected), the driver supports long columns when using a static cursor. This option is disabled by default. Enabling this option causes a performance penalty at the time of execution when reading long data.

"Configuring Database Parameters" on page 115

"Chapter 6, Configuring a Database" on page 103

Enable parent paths

By default, **Enable parent paths** is set to **no**. When **Enable parent paths** is set to **no**, a **FileSystemObject** object instantiated by an ASP application is limited to that application's defined directory. This is the most secure setting and is appropriate for most shared Web hosting environments.

When **Enable parent paths** is set to **yes**, the **FileSystemObject** object 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.

Caution

Changing **Enable parent paths** to **yes** can affect the security of your server. Before you change this setting, make sure that the 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.

Note

The **Enable parent paths** setting does not add any restrictions to executing Java code. For example, if you want to restrict Java code to access files within the application directory, the proper permissions should be in the bean.policy file.

See also:

"Configuring File System Access" on page 56

Environment Database Specific Variables

Oracle and Informix database servers require this additional environment information (with client). Ask your database administrator for more information about which settings to use.

See also:

"Configuring the Database Environment" on page 112

File size limit for blob column (bytes)

Specify the blob file-size limit in bytes. In most cases you should leave this set to the default.

Note

This value defaults to the `maxlongfieldlength` setting in the Sun ONE ASP configuration file, casp.cnfg, which specifies the maximum long field length in bytes. By default this value is 65535. If the data passed to a database exceeds this limit, ADO will throw an error. While it is recommended that you leave this set to the default, this value can be increased if necessary.

See also:

"Administering DBMS" on page 149

Host

This is the IP address of the SequeLink server. Ask your database administrator for this information.

See also:

"Configuring Database Parameters" on page 115

"Chapter 6, Configuring a Database" on page 103

HostName

This field requires different information for different databases:

- For Oracle databases, specify 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.
- For Informix databases, specify the name of the computer on which the Informix server resides.

See also:

"Configuring Database Parameters" on page 115

"Chapter 6, Configuring a Database" on page 103

Informix

This environment information is required to configure the ASP Server to connect to an Informix database. Ask your database administrator for more information about which settings to use.

See also:

"Setting Informix Environment Variables" on page 114

Inherit user security

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, especially if you are running Sun ONE Web Server. For more information about the security modes and their implications, see "Setting the Security Mode" on page 57.

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 httpd.conf file. This is the default security mode for Sun ONE ASP. This mode is available only for Sun ONE ASP running with the Apache Web server.

When **Inherit user security** is set to **no**, the ASP Server runs as root, unless a different user and group is specified in the Sun ONE 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 and group in casp.cnfg, as described in "Editing the Sun ONE ASP Configuration File" on page 517 (see the `[default machine]` section). This mode is available for both the Sun ONE and Apache Web servers.

See also:

"Changing ASP Server Settings" on page 37

"Securing the Server" on page 55

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 (for example, "A, b, C" would be sorted as "A, b, C").

See also:

"Configuring Database Parameters" on page 115

"Chapter 6, Configuring a Database" on page 103

IPAddress

This is the IP address for the DB2 database server.

See also:

"Configuring Database Parameters" on page 115

"Chapter 6, Configuring a Database" on page 103

Java VM Security Manager

When Sun ONE ASP Chili!Beans is enabled, you have the option of enabling the Java virtual machine (VM) Security Manager. 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 Java 2 Security documentation.

Key

Specify the authorization key for Sun ONE ASP Database Publisher or DBMS. The key unlocks the applications for use on the client side, and must be supplied to all users of the applications (the key is the same for all users).

For both tools the key is configured as "password" by default. A new key should be chosen as soon as possible.

See also:

"Chapter 7, Using Database Tools" on page 135

"Administering Database Publisher" on page 136

"Administering DBMS" on page 149

Locale

The language specified in this box sets the default locale identifier (LCID) and code page for the ASP Server.

See also:

"Configuring International Support" on page 43

"Changing ASP Server Settings" on page 37

"Developing International Applications" on page 212

Location

Specify this attribute only if the DB2 database is running on OS/390.

This is a path that specifies the DB2 location name. Use the name that was defined during the local DB2 installation.

See also:

"Configuring Database Parameters" on page 115

"Chapter 6, Configuring a Database" on page 103

Log file

In the **Log file** box, specify the path for application logging information. If this field is empty, no logging is performed.

"Administering Database Publisher" on page 136

"Database Publisher" on page 135

Logging file

To create the ADO log file and enable logging, specify the absolute path name of the ADO log file. You cannot use the name of a file that already exists in the same directory. If this box is empty, no logging is performed.

Caution

ADO logging should be used for diagnostic purposes only, and should not be enabled when running Sun ONE ASP on a production server.

See also:

"Enabling and Disabling ADO Logging" on page 133

LogonID

Specify the username required for accessing the database. If the username is not provided here, every connection string using this DSN must include the username. Ask your database administrator for this information.

See also:

"Configuring Database Parameters" on page 115

"Chapter 6, Configuring a Database" on page 103

Max. Transfer Size (Bytes)

If you select **Chili!Upload** on the **Components** page, the **Max. Transfer Size (Bytes)** box displays. This box specifies the maximum size per transfer (in bytes) that can be uploaded using the Chili!Upload component.

See also:

"Enabling SpicePack Components" on page 483

"Chapter 14, SpicePack Component Reference" on page 483

Number of threads

This setting specifies the maximum number of threads handled at the same time by a single ASP Server. The default is **5**.

This is an advanced setting. A maximum number of up to 20 threads is recommended. DO NOT set this to a number greater than 20.

See also:

"Configuring Multi-threading" on page 71

ODBC Drivers

Sun ONE ASP includes the ODBC drivers for a number of databases. The **ODBC Drivers** page displays the ODBC drivers installed for the listed database types. An ODBC driver is a module that enables a database to be accessed through ODBC (Open Database Connectivity). A separate driver is required for each type of database.

See also:

"Supported in This Release" on page 5

"Creating Database Connections (ASP Server)" on page 44

"Chapter 6, Configuring a Database" on page 103

"Connecting to a Database" on page 197

Oracle

This environment information is required to configure the ASP Server to connect to an Oracle database (with client). Ask your database administrator for more information about which settings to use.

See also:

"Setting Oracle Environment Variables" on page 113

Package

This is the package created by the DataDirect driver that reflects all 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.

Caution

Do not edit this package. The package is unique to a specific database.

See also:

"Configuring Database Parameters" on page 115

"Chapter 6, Configuring a Database" on page 103

Password

A password is required for connecting to a database. Ask your database administrator for this information. If you do not specify a password when configuring a system DSN, the password must be included in every connection string that uses the DSN.

Port

Specify the port on which the database server is configured to listen.

- For MySQL, the default is 3306.
- For PostgreSQL, the default is 5432.
- For SequeLink, the default is 19996.
- For Sybase, the default is 4100.

Note

For SequeLink, this is the port the SequeLink server is listening on.

See also:

"Configuring Database Parameters" on page 115

"Chapter 6, Configuring a Database" on page 103

PortNumber

Specify the port on which the database server is configured to listen. Ask your database administrator for this information. For Oracle (without client), the default is 1521.

See also:

"Chapter 6, Configuring a Database" on page 103

"Configuring Database Parameters" on page 115

ProcedureRetResults

When this option is enabled (the check box is selected), Oracle returns record sets from a stored procedure call. This option is enabled by default.

See also:

"Configuring Database Parameters" on page 115

"Chapter 6, Configuring a Database" on page 103

ReadOnly

When this option is enabled (the check box is selected), the database is treated as read-only. This option is disabled by default.

See also:

"Configuring Database Parameters" on page 115

"Chapter 6, Configuring a Database" on page 103

Requests per second

This field displays the number of requests per second currently being processed by the ASP Server

See also:

"Monitoring ASP Server Performance" on page 61

Script timeout

This specifies the amount of time the ASP Server waits for an individual ASP page to finish processing before canceling the request. The default is 90 seconds.

See also:

"Changing the Script Timeout Value" on page 68

"Changing ASP Server Settings" on page 37

Scripts buffering on

Yes enables scripts buffering, which means that the ASP Server processes an entire ASP page before returning its HTML output to the browser. This yields better server performance. When scripts buffering is disabled (the value is set to **no**), the ASP Server returns the HTML output for an ASP page to the browser incrementally, as

soon as the HTML is processed. This makes debugging easier. This setting is **yes** by default.

See also:

"Enabling Scripts Buffering" on page 66

"Changing ASP Server Settings" on page 37

Server

Specify the database server name or IP address. If this field is empty, the database server is assumed to be running on the local computer.

See also:

"Configuring Database Parameters" on page 115

"Chapter 6, Configuring a Database" on page 103

ServerDataSource

This is the name of the DSN configured on the SequeLink server. For more information about server-side configuration, see "Configuring SequeLink" on page 128.

See also:

"Configuring Database Parameters" on page 115

"Chapter 6, Configuring a Database" on page 103

ServerIPAddress

Specify the IP address of the Windows-based database server. Ask your database administrator for this information.

See also:

"Configuring Database Parameters" on page 115

"Chapter 6, Configuring a Database" on page 103

ServerName

- For Oracle databases, specify the TNS name as defined in the tnsnames.ora file by the Oracle client utility.
- For PostgreSQL and Sybase databases, specify the IP address of the database server.
- For Informix, specify the name of the database server as it appears in the sqlhosts file.

If this field is empty, the database server is assumed to be running on the local computer.

See also:

"Configuring Database Parameters" on page 115

"Chapter 6, Configuring a Database" on page 103

ServerPortNumber

Specify the port on which the Windows-based database server is configured to listen. Ask your database administrator for this information. The default is **1433**.

See also:

"Configuring Database Parameters" on page 115

"Chapter 6, Configuring a Database" on page 103

Session timeout

This specifies the amount of time the ASP server maintains session values for a user without receiving a page request. The default is 20 minutes.

See also:

"Changing the Session Timeout Value" on page 67

"Changing ASP Server Settings" on page 37

Settings

The Administration Console **Settings** page provides access to ASP Server configuration settings. When you change any of these settings, you must restart the ASP Server.

See also:

"Changing ASP Server Settings" on page 37

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.

See also:

"Configuring Database Parameters" on page 115

"Chapter 6, Configuring a Database" on page 103

TableType

Specify the default table type (Comma, Tab, Character, Fixed, or Stream). The Text driver supports five table types: comma-separated, tab-separated, character-separated, fixed length, and stream. The default table type is used when creating a new table, and opening an undefined table.

See also:

"Configuring Database Parameters" on page 115

"Chapter 6, Configuring a Database" on page 103

TcpPort

This is the port on which the DB2 database server is configured to listen. Ask your database administrator for this information.

See also:

"Configuring Database Parameters" on page 115

Total errors received

This field displays the number of ASP Server errors.

See also:

"Monitoring ASP Server Performance" on page 61

Total memory in use

This field displays the system memory currently being used by the ASP Server.

See also:

"Monitoring ASP Server Performance" on page 61

"Optimizing ASP Server Performance" on page 66

Total requests

This field displays the total number of requests processed since the ASP Server was started.

See also:

"Monitoring ASP Server Performance" on page 61

Uptime

This field displays the length of time the ASP Server has been running since the last restart.

See also:

"Monitoring ASP Server Performance" on page 61

UseCursorLib

When this option is enabled (the check box is selected), ODBC Manager cursor support overrides the ODBC driver support, enabling scrollable cursors not supported by the ODBC driver. This option is enabled by default.

See also:

"Configuring Database Parameters" on page 115

"Chapter 6, Configuring a Database" on page 103

User

Specify the username required for accessing the database. If the username is not provided here, every connection string using this DSN must include the username. Ask your database administrator for this information.

See also:

"Configuring Database Parameters" on page 115

"Chapter 6, Configuring a Database" on page 103

User Classpath

If the **Chili!Beans** box is selected on the **Components** page, the **User Classpath** field also displays. Use this field to specify a classpath other than the default.

See also:

"Chapter 12, Chili!Beans Component Reference" on page 465

View Logs

The Administration Console **View Logs** page displays the ASP errors log file and ASP Server Diagnostics. Logging must enabled for logging data to be displayed, as described in "Enabling ASP Errors Logging" on page 63.

See also:

"Viewing the ASP Errors Log" on page 64

Virtual Hosts

Virtual hosts (called virtual servers on Sun ONE Web Server) is a feature that enables a Web server to service multiple hostnames. Sun ONE ASP automatically processes ASP applications for any virtual host defined on the Web server.

Settings to enable or disable ASP processing for individual virtual hosts are accessed on the **Web Server** tab of the Administration Console **Server Management** page.

See also:

"Enabling ASP for a Virtual Host" on page 54

"Defining Applications in a Shared Environment" on page 74

Index