

THEO+Server™

User's Guide and Reference



First Edition: August, 1998

Copyright © 1998 by THEOS Software Corporation.

All rights reserved.

The software described in this document is furnished under a license agreement or nondisclosure agreement. The software may be used only in accordance with the terms of the agreement. Information in this document is subject to change. No part of this manual may be reproduced, transmitted, transcribed or translated into any language in any form by any means without the written permission of THEOS Software Corporation. Printed in the United States of America.

THEOS Software Corporation
1801 Oakland Boulevard, Suite 315
Walnut Creek, CA 94596-7000

Telephone: 925 935-1118
Fax: 925 935-1177
E-mail: sales@theos-software.com
WWW: <http://www.theos-software.com>

THEOS® and the THEOS logo are registered trademarks and THEO+Mail™, THEO+Net™ and THEO+Server™ are trademarks of THEOS Software Corporation. Internet Explorer®, Microsoft®, Windows®, Windows 95® and Windows NT® are registered trademarks of Microsoft Corporation. Netscape Navigator® is a registered trademark of Netscape Communications Corporation.

Documentation by Computer Printed Words.

Table of Contents

Introduction	9
--------------------	---

Part I: Installation and Setup

1 Installation	17
Your THEO+Server Package	17
Installation Procedures	17
Acquiring an Updated Copy of THEO+Server	19
Using THEO+Net To Acquire Updated Copy	19
2 THEO+Server Quick Setup	21
THEOS FTP Server Quick Setup	21
Creating a Special FTP Directory	21
Creating a User Definition	22
Starting the Server	23
THEOS HTTP Server Quick Setup	24
The HTTP Server Name	24
Starting the Server	25
3 Setup Command	27
Setup FTP	29
General Options	30
Log Options	32
Messages	34
Access	35
Groups/Users	37
Group Definition	37
User Definition	41
Setup HTTP	44
Service Options	45
Directory Setup	48
Virtual Directories	49
User Security	50
Network Security	51
File Associations	52
Cancel	53
Exit	53
4 Server Security	55
Controlling Access to the Servers	55
Allow IP and Deny IP Lists Limit Access to the Server	57

Controlling Access to the Server's File System	59
Virtual Directories Limit Access to Directories	60
Logging User Activities	61
FTP Server Logging	61
HTTP Server Logging	62

Part II: THEOS FTP Server

5 Using the FTP Server	67
Starting and Stopping the Server	67
Starting the FTP Server Automatically	67
Starting and Stopping the FTP Server Manually	67
Using the FTP Server with Dial-Up Networking	68
Limiting the Number Of Connections	68
Controlling Access to Your File System	69
Home Directories	70
Virtual Directories	71
Access Permissions	72
Defining Users	74
Using Groups	74
Adding and Maintaining Users	76
Using Custom Messages	78
Sign-on and Sign-off Messages	79
Login Message	80
Directory Change Message	82
Anonymous Users	84
Testing the Server Configuration	86

Part III: THEOS HTTP Server

6 Using the Web Server	91
What is an HTTP or Web Server?	91
Starting and Stopping the Server	92
Starting the HTTP Server Automatically	92
Starting and Stopping the HTTP Server Manually	92
Startup Environment Variables	93
Using the HTTP Server with Dial-Up Networking	93
Limiting Access by IP Number or Domain Name	93
Controlling Access to Your File System	94
Virtual Directories	95
Controlling Access to Specific Directories	96

Controlling Search Engine and Robot Access to Web Site	97
Client Access	100
Default Documents	100
Directory Browsing	101
Customizing the Directory Browse View	102
Customizing Common Resources	105
File Icons	105
Error Message Documents	106
7 Dynamic Web Pages	109
Server Side Includes (SSI)	110
Server-Side-Include Format	110
Active Server Pages (ASP)	112
Common Gateway Interface (CGI)	113
CGI Program Location	114
Server, HTTP and Environment Variables	115
Limitations	115
Client-Side JavaScript	116
Validating Form Input	117
Embedding JavaScript in HTML	117
Using the SCRIPT Tag	117
Specifying a File as JavaScript Source	118
Hiding Scripts within Comment Tags	118
8 Getting Information	119
Information from the Server	119
Server Variables in SSI	119
Server Variables in CGI	119
Server Variables in ASP	120
Types of Information Received from Client	120
Browser Identification	121
Query String	122
Query Strings and SSI	122
Query Strings and CGI	122
Query Strings and ASP	123
Cookies	123
Cookie Values and SSI	124
Cookie Values and CGI	124
Cookie Values and ASP	125
Form Data	125
Form Data and CGI	125
Form Data and ASP	126

Appendices

A	Contacting THEOS	127
B	THEO+Server Support Files	129
C	FTP Log File	131
D	HTTP Log File	133
E	FTP Message Variables	135
F	SSI Directives	137
	SSI Syntax	137
	SSI Directives	138
	#config	139
	#counter	144
	#echo	147
	#exec	150
	#lastmod	152
	#size	154
	#include	155
	#nossi	156
G	MultiUser Basic Language CGI API Functions	157
	CGI Functions	157
	CGI.INIT, CGI.BEGIN, CGI.FINISH, CGI.PUT.COOKIE	159
	CGI.ERROR, CGI.LOCATION, CGI.STATUS	164
	FN.CGI.BROWSER\$	166
	FN.CGI.GET functions	168
	FN.FORM functions	170
	FN.HTML functions	177
	FN.JAVA functions	184
	FN.TABLE functions	185
	FN.URL.DECODE\$, FN.URL.ENCODER\$	188
H	ASP Reference	189
	General Syntax	189
	White Space and Carriage Returns	190
	Casemode	190
	Comments	191
	Expressions	191
	Constants	191
	Variable Names	192

Integer.....	192
Boolean.....	192
Date and Time.....	192
String.....	193
Operators.....	193
Assignment.....	196
AtEndOfLine.....	197
AtEndOfStream.....	197
Close.....	198
Column.....	199
Line.....	199
CreateTextFile.....	200
Do.....	202
FileSystemObject.....	204
For.....	205
Functions.....	207
If.....	214
MSWC.BrowserType.....	216
OpenTextFile.....	218
Read.....	220
Request.....	221
Response.....	224
Select Case.....	226
Server.....	228
Set.....	230
Skip.....	231
TextStream.....	232
While.....	234
Write.....	235
I CGITEST Command.....	237
Glossary of Terms.....	239
Index.....	245

Introduction

Welcome to THEO+Server Version 1, a Plus Pak for the THEOS 32 Version 4 operating system. THEO+Server is composed of two related but separate network servers.

- ▶ **FTP Server**

Allows network clients to transfer files from and to the server computer using the File Transfer Protocol.

- ▶ **HTTP Server**

Allows network clients to retrieve documents and post form data fields from and to the server computer using the Hypertext Transfer Protocol.

■ **THEOS FTP Server**

The FTP server allows a THEOS-based multiuser system to be a file server to other systems on the local network or on the Internet. FTP refers to File Transfer Protocol which is a standard file transfer protocol used to send and receive data and text files between a server and a client.

- ▶ Uses FTP defined in RFC 959.
- ▶ Uses any network connection available with THEO+Net including dedicated LAN, dedicated Internet or dial-up connection to an ISP.
- ▶ Easy installation and configuration.
- ▶ Security provided with allow/deny lists for IP addresses, password-protected directories and virtual directories.
- ▶ Connections to multiple clients limited only by number of tasks available on system and by server configuration.
- ▶ Anonymous users access can be enabled or disabled. Number of simultaneous anonymous users can also be control by the server configuration.
- ▶ Ability to log all requests from a user agent for each document and for each posting of form data.
- ▶ Log files can be automatically rotated on a daily, weekly or monthly basis.

- ▶ Virtual directories to control network access to the file system on the server's machine.
- ▶ Login and logout messages can be customized.
- ▶ Directory change message can be customized and may be specific to each directory.
- ▶ Each user account specifies exactly which directories the user may access. Each "virtual directory" specifies exactly what type of access the user has: read, write, delete, list, *etc.*

■ **THEOS HTTP Server**

The HTTP server, sometimes referred to as the Web Server, allows a THEOS-based multiuser system to be a network server for web documents on a local network or on the Internet. HTTP refers to the HyperText Transfer Protocol which is a standard data transfer protocol used to send documents and other data between a server and a client such as a web browser.

- ▶ Uses HTTP version 1.0 defined in RFC 1945.
- ▶ Uses any network connection available with THEO+Net including dedicated LAN, dedicated Internet or dial-up connection to an ISP.
- ▶ Easy installation and configuration.
- ▶ Security provided with allow/deny lists for IP addresses, password-protected directories and virtual directories.
- ▶ Multiple connections allowed between server and each client (standard feature of all HTTP servers).
- ▶ Multiple connections to multiple clients limited only by number of tasks available on system.
- ▶ Server Side Includes (SSI) supported with built-in counters.
- ▶ Common Gateway Interface (CGI) version 1.1 supported.
- ▶ CGI ToolKit for developing your own CGI applications.
- ▶ Active Server Page (ASP) support for delivering dynamic documents to the client including if-then-else conditional HTML code.
- ▶ Ability to log all requests from a user agent for each document and for each posting of form data.
- ▶ Log files can be automatically rotated on a daily, weekly or monthly basis.
- ▶ Virtual directories to control network access to the file system on the server's machine.

- ▶ Directory browsing display supporting customizable heading and footing text and descriptions for each file in the directory.
- ▶ Directory browsing display can be disallowed.
- ▶ Default literal text used for directory browsing can be customized by translation to national language.
- ▶ Example documents provided to illustrate some of the features of SSI, CGI, ASP and JavaScript enabled documents.

■ **Hardware Requirements**

THEO+Server has the following minimum hardware requirements:

- ▶ 8 Mb RAM
- ▶ 10 Mb available disk space
- ▶ IEEE 802.3 Ethernet medium (cable)
- ▶ Ethernet Network Interface Card (NIC)
- ▶ Multiple computers and/or access to the Internet

The actual amount of RAM and disk space required depends upon the services started, number of sockets enabled, *etc.*

■ **Software Requirements**

THEO+Server requires THEOS 32 Version 4, patch level 40275 or above and THEO+Net Version 4.

■ **About This Manual**

This manual is designed to show you how to install, configure and use THEO+Server Version 1 on an existing THEOS 32 Version 4 system. It is divided into five parts:

- ▶ **Part I** “Installation and Setup”

Gives a general overview of networking concepts, terminology and components with emphasis on those applying to THEO+Net.

- ▶ **Part II** “Configuring the THEOS FTP Server”

Step-by-step instructions for installing and configuring the software.

- ▶ **Part III** “Configuring the THEOS HTTP Server”

All commands and setup programs are described in detail.

- ▶ **Appendices**

Support information, [Glossary of Terms](#), and an [Index](#).

■ Documentation Conventions

Throughout this manual, syntax is used that looks like:


```
NET
NET START server-name
NET function parameters
FTP filename (FILE options)
```

These symbols are used to show the correct syntax for typing the command.

BOLD Words and characters shown capitalized and in boldface must be entered exactly as shown.

italics Italicized words show parameters whose value is supplied by you.

FILELIST Underlined portions of a word indicate the minimum spelling allowed for abbreviations of the word.

 Specific keys on the keyboard are indicated by icons of the key. These standards are used in the formal definitions of the syntax of a command or feature and in the descriptive text of the feature. The descriptive text also uses the following typographic conventions to identify the various items described.

File names File names that appear in text are always shown in small caps. For instance, the SYSTEM.THEOS32.DEV NAMES file.

Definitions A significant word or term that is being defined in the text is shown in boldface italics. For instance: The ***command name*** is the name of the program that you want to execute.

Literal text Text appearing in an example, that is also referred to in the text description of the example, is shown with a different typeface.

Part I ---

Installation and Setup

1 Installation

This section assumes that you have a working computer system, with THEOS 32 Version 4 installed and configured, and with THEO+Net installed and configured.

■ Your THEO+Server Package

The THEO+Server package that you received consists of a floppy diskette, an *authorization code* and this manual.

The authorization code is supplied on paper. **This is the only copy that you will receive.** Please keep this document with your operating system diskettes and make several copies. Replacement authorization codes can be obtained from your THEOS reseller, but that takes time and there may be a charge.

■ Installation Procedures

To install THEO+Server Version 1, follow these steps:

1. Log onto the “SYSTEM” account and make sure you are at the CSI prompt.
2. If either the FTP server or the HTTP server is started, stop them:

```
>net stop ftp
```


```
>net stop http
```

3. Load the diskette provided with THEO+Server into your diskette drive.
4. Enter the command:

```
>install
```

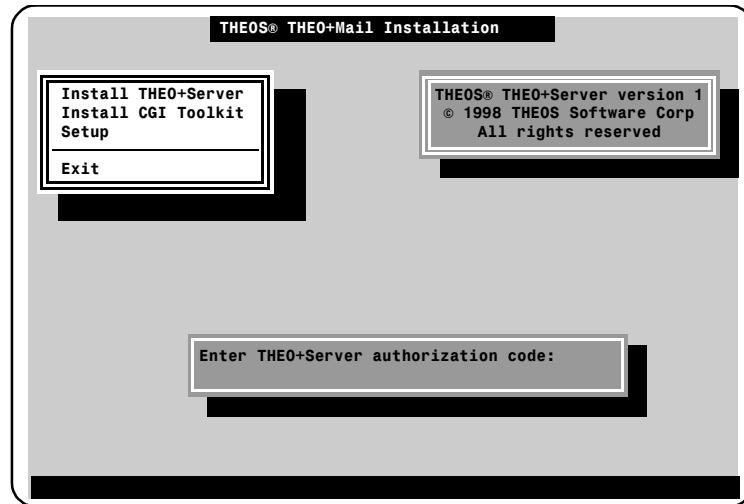
If the diskette drive you are using is not the normal, default “F” drive, specify the drive code letter after the “install” command name.

For instance, if you are using the “G” drive, enter the command:

```
>install g 
```

This program loads and executes the installation program for THEO+Server from the attached drive.

5. The THEO+Server Version 1 installation screen displays and prompts you to enter the authorization code, which was supplied as a document with this product.



6. Enter the authorization code in the box and press **Enter ↵**. It can be entered with uppercase or lowercase letters.

If you enter an invalid authorization code, the message: “Incorrect access code!” is displayed. You are given three chances to enter a correct code. If a valid authorization code is not entered on the third attempt, the installation process exits.

7. If the code is accepted, you can select one of the options in the menu displayed in the upper-left portion of the screen.
 - ▶ The “Install THEO+Server” item installs both the FTP server and the HTTP server onto your system.
 - ▶ The “Install CGI Toolkit” item installs the CGI function library and the example programs onto your system.
 - ▶ “Setup” invokes the `SETUP SERVER` command which was one of the programs installed by the “Install THEO+Server” process. The servers can also be configured after installation. Refer to Chapter 3 “[Setup Command](#)” starting on page [27](#).

This completes the installation of THEO+Server Version 1.

■ **Acquiring an Updated Copy of THEO+Server**

A system with THEO+Server already installed on it may be updated to a later version by obtaining the most current version from THEOS Software Corporation (resellers) or by downloading the current version from the THEOS Web site (resellers or end users). The THEOS Web site contains the latest version of most software provided by THEOS Software Corporation.

The latest version of the commercial release of THEO+Server can be found by going to the “Software Library” link in the home page

<http://www.theos-software.com>

Be sure to view the “Installation instructions” link in the download directory entry for THEO+Server. A software portal and a matching authorization code are required to install THEO+Server onto a computer, even when the software was downloaded from the Web site.

The THEOS Web site may also allow you to download a “beta” version of THEO+Server. A **beta version** of software is a prerelease version and is not intended to be installed on a production system. A beta version may contain errors that could affect data integrity or operation of your system.

■ **Using THEO+Net To Acquire Updated Copy**

If you already have a version of THEO+Net on your system and it has Internet access, you may get the current version of THEO+Server from the THEOS FTP site using the FTP client included with THEO+Net.

```
>ftp ftp.theos-software.com
possible proxy connection messages
-----
Welcome to THEOS Software Corporation FTP site.
Anonymous user logged in.
-----

FTP? bin
Using Binary mode to transfer files.

FTP? cd theos/theoserv
CWD command successful.

FTP? ls
theoserv.zip

FTP? get theoserv.zip
Receiving theoserv.zip (1 of 1)
422,370 bytes received in 143.6 seconds (2.94 Kbytes/sec)
```

```
FTP? bye
Goodbye
```

```
>
```

The actual files downloaded will be the current version of the THEO+Server product. They may have a different name and patch level than the file indicated.

The specific name of the compressed file that was actually downloaded must be used when uncompressing it.

```
>unzip theoserv.zip
Archive: /TEMP/THEOSERV.ZIP:S
  inflating: THEOSERV.IMG
```

Copy the image file to floppy diskettes.

```
>attach i image1

Enter IMAGE file name (or ESC to exit): THEOSERV.IMG

>backup i f (noask

>attach i
```

Use this diskette to install the new version of THEO+Server or use the image drive for the installation:

```
>attach i image1

Enter IMAGE file name (or ESC to exit): THEOSERV.IMG

>install i
```

2 THEO+Server Quick Setup

This chapter describes the minimum setup that must be performed to begin using the FTP and HTTP servers. Read Chapter 3 “[Setup Command](#)” on page 27, which describes all of the fields that you may configure for your servers.

■ THEOS FTP Server Quick Setup

There is very little that must be done to begin using the FTP server. The settings defined in the default configuration are adequate for most sites, although you may want to change them to make the server more customized for your specific operations. Refer to “[Setup FTP](#),” starting on page 29 for information about all of the settings that you can change.

To begin using the FTP server, the minimum that you must do is:

- ▶ Create user definitions
- ▶ Start the server

An additional step that you may do will help you in controlling access to the files on your system. Although not necessary, creating a new directory in the SYSTEM account just for FTP access greatly simplifies the control of remote access to your file system.

Of course, your system must be accessible to users wishing to connect to your server. That is, they must have TCP/IP network access to your system.

■ Creating a Special FTP Directory

On your SYSTEM account, create a directory structure similar to:

```
/ftp
  /incoming
  /download
```

This directory doesn’t necessarily have to be on the system account, but it makes it easier if it is. You may use any names that you want, but the `/ftp` and `/ftp/incoming` directory names are the conventional ones used on many FTP servers.

The `/ftp/incoming` directory is used as a location where FTP users can upload files to your system. The `/ftp/download` directory will contain files

that you want users to be able to download. When you define specific users, you can give them access to additional directories.

■ Creating a User Definition

The FTP user account defines the user name and password used to access the server and it defines the files that may be accessed when that user logs onto the server.

For minimal usage, you should create at least two user definitions, one for a specific user and another for the special user name of “anonymous.” After you have created and used these user accounts, you may want to create one or more group definitions so that it will be easier to create and maintain the user accounts for many similar users.

To create the “anonymous” user, use the “Users” item of the “Groups/ Users” menu of the Setup Server command, FTP menu item.

The screenshot shows a terminal window titled 'Users'. It displays the configuration for the 'Anonymous' user. The fields are as follows:

User Name: Anonymous		
Group Name:		
Home Directory: /ftp:s		
Login Msg Text:		
Hide Hidden: NO		
Password:		
VIRTUAL	PHYSICAL	PERMISSIONS
/	/ftp:s	RLS
/incoming	/ftp/incoming:s	WLCS

Fill in the fields at the top of the screen as shown above. To create the virtual directories shown, press the **[Ins]** key for each directory and fill in the values shown below:

The screenshot shows a terminal window titled 'Add Directory'. It displays the configuration for virtual directories. The fields are as follows:

Virtual	/
Physical	/ftp:s
Read access	YES
Write access	YES
Listing	YES
Create	YES
Delete	YES
Modify	YES
Subdirectory	YES

Add Directory	
Virtual	/incoming
Physical	/ftp/incoming:s
Read access	NO
Write access	YES
Listing	NO
Create	YES
Delete	NO
Modify	NO
Subdirectory	YES

The first virtual directory gives anonymous users full access to your /ftp directory unless this access permission is overridden by another virtual directory definition. The second virtual directory restricts their access to the /ftp/incoming directory so that they will be able to upload files to the directory but they will not be able to download from the /ftp/incoming directory nor will they be able to list or delete files or subdirectories in that directory.

Remember to use the **[F10]** key to save each screen. Pressing **[Esc]** instead of the **[F10]** key will abandon the changes or additions without saving them.

Perform similar operations to create a normal user definition. However, for normal users you will want to specify a password.

■ Starting the Server

You can start the FTP server “manually” by using the command:

```
>net start ftpserver
```

You may also configure the server to start every time that your system is booted. To do this, change the field “Auto Start” in the “General” settings of the Setup FTPD command to “YES.”

General	
Port:	21
User Probes:	3
Max User:	10
Max Anon:	10
User Timeout:	60
Anon Timeout:	15
Auto Start:	YES
Check Anon Pass:	NO
Email:	
Lowercase Dir:	NO
Dir Change Msg:	

Use the **[F10]** key to save your change. The next time that your system is booted, the FTP server will be started.

■ THEOS HTTP Server Quick Setup

Like the FTP server, there is little that must be done to begin using the HTTP server because the settings defined in the default configuration are suitable for most sites. You may want to change these default settings to make the server more customized for your specific operation. Refer to “[Setup HTTP](#),” starting on page 44 for information about all of the settings that you can change.

To begin using the HTTP server, the minimum that you must do is:

- ▶ Specify your server’s name
- ▶ Start the server

You may wish to implement some of the optional features of the server such as virtual directories, network and user security. Refer to Chapter 6 “[Using the Web Server](#)” starting on page 91 for descriptions of these features.

You do not have to create a special directory for the server’s usage because one was created for you during the installation of THEO+Server. This directory is named /www:s and initially is empty. Before the HTTP server can be useful, one or more documents should be created in or moved to this directory.

Of course, your system must be accessible to users wishing to connect to your server. That is, they must have TCP/IP network access to your system, either on a LAN or via the Internet.

■ The HTTP Server Name

HTTP servers such as the THEOS HTTP server create MIME headers for each document delivered to a **user agent** (web browser). The server name specified in the HTTP configuration is used by the server to provide some of the information that is required in these document headers.

You must specify a server name in the “[Setup HTTP](#)” command, “[Service Options](#)” settings, in the “[Server Name](#)” field. If your system is registered with InterNIC or the ISP providing your Internet access, use the registered name. For instance, if your registered domain is “mycompany.com” and the machine is “www,” then specify a server name of “www.mycompany.com.”

If your system is not registered use your IP address for the server name. For instance, “192.168.123.200.”

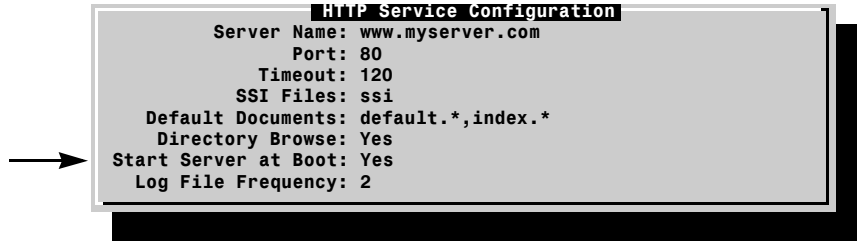
If your system is not registered and it is used only for LAN access, you can use any name that you want for the server name.

■ Starting the Server

You can start the HTTP server “manually” by using the command:

```
>net start webserver
```

You may also configure the server to start every time that your system is booted. To do this, change the field “[Start Server at Boot](#)” in the “[Service Options](#)” settings of the “[Setup HTTP](#)” command to “YES.”



Use the **[F10]** key to save your change. The next time that your system is booted, the HTTP server will be started.

3 Setup Command

The Setup command provides a single command to configure and initialize the major components of THEOS, various types of devices and optional components such as THEO+Server.

1	<u>SETUP</u>				
2	<u>SETUP</u>	<i>function</i>			
<hr/>					
<i>function</i>	»	ACCOUNT	DISK	FLOPPY	SERVER
		COLOR	DOS	FTP	SIO
		CRT	EMAIL	NET	SYSGEN
		DIGIXE	FAX	NETUSER	TELNET

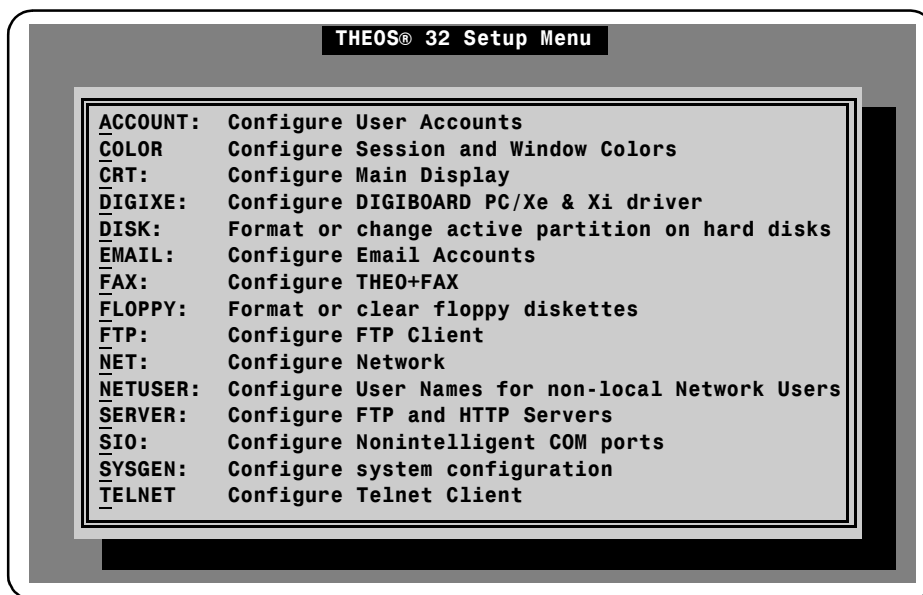
- Operation

Mode 1—Invokes Setup in its menu mode. See “[Setup Menu](#)” below.

Mode 2—Bypassing the Setup menu, the *function* screen is displayed. Refer to “[Functions](#)” on page 28.
- Setup Menu

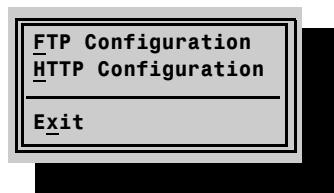
When Setup is invoked with [Mode 1](#), the Setup Menu is displayed. This menu is dynamic because only those components installed on your system are presented in the menu. For instance, if you do not have the THEO+Fax software installed on the system, the FAX menu item is not offered.

There may be additional items displayed on the menu, depending upon any add-on products that you may have installed on your system.

**Functions**

The following functions are available only after THEO+Server is installed on the system.

SERVER Invokes the THEO+Server Configuration screen.

THEO+Server Configuration

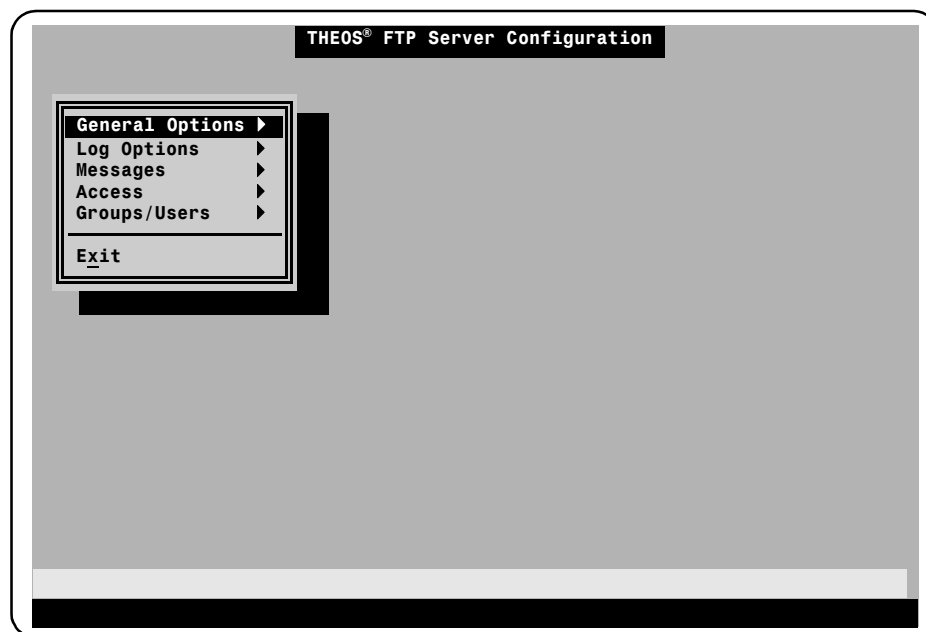
FTP Invokes the THEOS FTP Server Configuration screen described on the following pages. It configures the network File Transfer Protocol server's general operation and creates and maintains group and user account configurations.

HTTP Invokes the THEOS HTTP Server Configuration screen described on page 44. It configures the network HyperText Transfer Protocol server's general operation and creates and maintains user account configurations.

For other menu items in this command, refer to the **Setup** command description in the *THEOS System Reference* manual or to the specific Plus Pak manual such as *THEO+Net Installation and Reference Manual*.

Setup FTP

The “THEOS FTP Server Configuration” menu maintains the configuration of the server and maintains groups and user account configurations for users accessing the server.



Use the normal menu selection keys to select the desired item.

General Options. Configures the global settings for the FTP server (see page 30).

Log Options. Defines the log file name and the events that are logged to this file (see page 32).

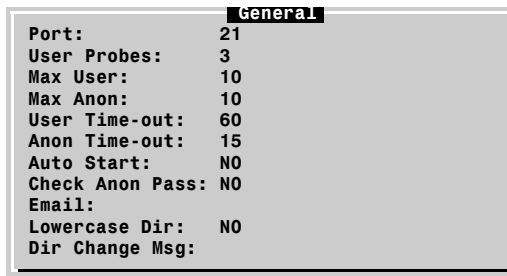
Messages. Defines the signon and signoff messages displayed to users connecting to the FTP server (see page 34).

Access. Configures the lists of IP addresses and domain names that are allowed to connect to or are denied access to this FTP server (see page 35).

Groups/Users. Configures group profiles and user accounts for users connecting to this FTP server (see page 37).

■ General Options

This screen configures the basic operating properties of the FTP server. Any changes to these settings will not take effect until the next time that the server is started.



General	
Port:	21
User Probes:	3
Max User:	10
Max Anon:	10
User Time-out:	60
Anon Time-out:	15
Auto Start:	NO
Check Anon Pass:	NO
Email:	
Lowercase Dir:	NO
Dir Change Msg:	

Port This is the port number that the server listens to for incoming requests. The default port number is 21 and, in most situations, should be used.

You may use any port number that is unused by other servers on your system. However, when a port number other than 21 is used, you will have to tell all users who want to access your server what that port number is. FTP client programs will use port number 21 unless they are specially configured.

A list of the well-known services and port numbers used by those services can be found in the file `SYSTEM.TEOS32.SERVICES`.

User Probes Specifies the maximum number of times that an FTP client can attempt to connect with an invalid account or password. With a default value of 3, this means that the user is allowed three connection attempts before they are automatically disconnected.

Max User Defines the maximum number of users that can be connected to this server at any one time. This number limits the total number of regular users and anonymous users that may connect.

Max Anon Defines the maximum number of anonymous users that can be connected to this server at any one time. The number of anonymous users that may connect at one time is limited to the smaller of the value specified here and the maximum users specified in the previous field minus the number of users already connected.

User Time-out Specifies the maximum number of minutes that a connected user may be idle before they are automatically disconnected. The default value is 60 minutes (one hour).

Anon Time-out Specifies the maximum number of minutes that a connected anonymous user may be idle before they are automatically disconnected. The default value is 15 minutes.

Auto Start A “YES” value will start the FTP server each time that this system is booted. A “NO” value will not start the server automatically.

You can manually start and stop the FTP server with the NET START command:

```
>net start ftpserver
```

```
>net stop ftpserver
```

Check Anon Pass A “YES” value requires that anonymous logins must enter a password and the password entered is validated for proper syntax. When enabled, anonymous passwords must appear to be an e-mail account in the form:

```
user@mail.server
```

That is, it must contain one commercial at symbol (@) and a period in the domain name portion. The password is not actually checked to determine if it is a valid e-mail account.

When this field is set to “NO,” users logging in as “anonymous” may enter any password that they want or they may enter no password.

Email Defines the value for the message variable %help. By convention, this value should be an e-mail address for the account used by people needing assistance in using this FTP server. However, it may be any text that you want to appear when the %help variable is used.

For information about message variables, refer to Appendix E: “[FTP Message Variables](#)” starting on page 135.

Lowercase Dir When set to “YES,” directory names are forced to lowercase characters before being given to the client application. A “NO” value will use the default, uppercase names for directories and file names.

Dir Change Msg Specify the path and name of the text file containing the directory changed message. This file must be accessible by the server. That is, it should not be read protected.

If a path is not specified, or if the path is relative (doesn't start with a "/" character), the path is relative to the current working directory requested by the FTP client application.

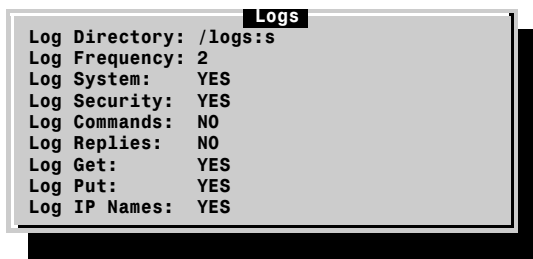
The directory changed text may use message variables, such as the %dir variable. Appendix E: "[FTP Message Variables](#)" describes the message variables available. Refer to "[Directory Change Message](#)" on page 82 for more information.

You may enter an **[Esc]** at any time to abandon any changes that you have made. Press **[F10]** to save your changes and return to the "THEOS FTP SERVER Configuration" menu.

Remember, changes to these settings will not take effect until the next time that the server is started. If it is currently running, it must be stopped and then restarted for these new settings to take effect.

■ Log Options

This configuration screen controls whether or not events are logged to an external file and which events are logged. Any changes to these settings will not take effect until the next time that the server is started.



Log Directory Defines the directory used for FTP log files.

The directory specified here is not created automatically. If the directory does not exist then no logging is performed. When it does exist when the FTP server is started, the appropriate log file is created or appended to if it already exists. Refer to the [Log Frequency](#) field described next for information about the names used for FTP log files.

Refer to Appendix C: “[FTP Log File](#)” starting on page 131 for a description of the format of records created in the FTP log file.

Log Frequency The code in this field controls whether the FTP log file is created and the frequency that it is “rotated.”

- 0 Do not create or rotate the FTP log file.
- 1 Create an FTP log file but do not rotate it. Log file name is always FTP.LOG.
- 2 Create an FTP log file and rotate the log file daily. Log file name is FTyymmdd.LOG where *yy* is the year number, *mm* is the month number and *dd* is the day of the month number.
- 3 Create an FTP log file and rotate the log file weekly, on Mondays. Log file name is FTyymmdd.LOG where *yy* is the year number, *mm* is the month number and *dd* is the day of the month number for the Monday of this week.
- 4 Create an FTP log file and rotate the log file on the 1st of each month. Log file name is FTyymm01.LOG where *yy* is the year number and *mm* is the month number.

The location of the log file is defined by the [Log Directory](#) field described above.

When the log [Log Frequency](#) code is 2, 3 or 4, the log file is rotated at the indicated frequency. The actual rotation occurs when a new transaction is to be created in the log. If the period has elapsed and a new file has not been created yet, the current file is closed and a new file is opened.

For instance, the frequency code is 3 and the last activity was on Wednesday, May 27, 1998. The current log file is named FT980525.LOG which is the date for the Monday of that week. If a new entry is to be added on Thursday, June 4, 1998 the current file is closed and a new file is started named FT980601.LOG, which is the date for the Monday of that week.

Log System A “YES” indicates that major events such as connecting, disconnecting, directory change are logged to the file.

Log Security A “YES” indicates that all login attempts and failures are logged to the file.

Log Commands A “YES” indicates that all commands from the FTP client are logged. This option should only be used for debugging purposes because it generates a large log file.

Log Replies A “YES” indicates that all replies to commands from the FTP client are logged. This option should only be used for debugging purposes because it generates a large log file.

Log Get A “YES” will log every file downloaded to the FTP client.

Log Put A “YES” logs every file uploaded from the FTP client.

Log IP Names A “YES” means that a user’s IP number is looked up in the DNS server and the domain name is logged when found.

You may enter an **[Esc]** at any time to abandon any changes that you have made. Press **[F10]** to save your changes and return to the “THEOS FTP Server Configuration” menu.

Remember, changes to these settings will not take effect until the next time that the server is started. If it is currently running, it must be stopped and then restarted for these new settings to take effect.

Refer to Appendix C: “[FTP Log File](#)” starting on page [131](#) for a description of the log file format.

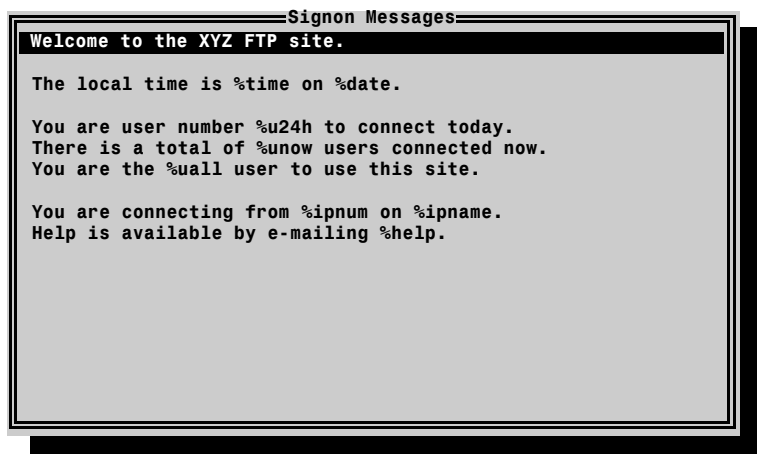
■ Messages

Two sets of messages can be defined with this menu. The signon messages are output to a user when they first connect to the server, the signoff messages are output when they disconnect. A special user login message can be defined for each user. Those user-specific messages are defined in the Group and User maintenance screens, described on pages [38](#) and [42](#).



Signon Messages This item allows you to maintain the messages output when a user first connects but before they login as a user.

Signoff Messages This item allows you to maintain the messages output when a user disconnects.



Both signon and signoff messages are maintained with an edit window. Type the lines of text that you want displayed during signon or signoff. The normal editing keys available in any edit window are available here to assist you in editing the text.

These messages may contain variable name references. For information about the variable names available, refer to Appendix E: “[FTP Message Variables](#)” starting on page 135.

■ Access

This menu allows you to define and maintain lists of IP address or names that are allowed or denied access to this FTP server. Any changes to these settings will not take effect until the next time that the server is started.



The allow and deny lists are lists of IP addresses that the FTP Server uses to determine whether or not a remote site is allowed to connect to the server. The allow list is a list of addresses, address ranges and address subnets that are specifically allowed to connect to the server. The deny list is a similar type of list of addresses that are specifically not allowed to connect to the server.

Both lists of IP addresses are used in sequence. For instance, if a site matches one of the entries in the allow list and also matches one of the entries in the deny list, the site is not allowed to connect.

A list is only checked if there is at least one entry defined in the list. For instance, if there are no entries in the allow list but there is an entry in the deny list, then all sites are allowed to connect except those listed in the deny list.

For further information about the usage of allow and deny lists, refer to the chapter “Network Security” in the *THEO+Net Installation and Reference* manual. Although the two sets of lists (one for the FTP server and another for the Login server) are separate, both operate in the same manner.

An entry in the allow or deny list may be specified in one of several forms:

- ▶ The host name as defined in the file `SYSTEM.THEOS32.HOSTS`. This file is maintained by the Host Names Database screen in the **Setup NET** command.

my-company

- ▶ The domain name as defined by the Domain Name Service specified in DNS Servers screen in the **Setup NET** command.

myisp.com

- ▶ The dotted IP address for a site.

207.21.75.100

- ▶ The dotted subnet IP address for a group of sites.

192.168.*.*

With this syntax, all sites whose IP address matches the wild-card specification are treated as a match for the list.

- ▶ A range of IP addresses or subnet IP address.

192.168.50.1-48

This syntax specifies that all sites will match if their IP address is 192.168.50.1, 192.168.50.2, ..., 192.168.50.48.

The CIDR form may also be used for specifying a range of IP addresses:

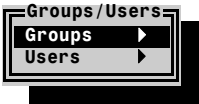
192.168.50.1/48

You may enter an **[Esc]** at any time to abandon any changes that you have made. Press **[F10]** to save your changes and return to the “THEOS FTP SERVER Configuration” menu.

Remember, changes to these settings will not take effect until the next time that the server is started. If it is currently running, it must be stopped and then restarted for these new settings to take effect.

■ Groups/Users

Group profiles and user accounts are created and maintained with this menu item.



Groups A group is a profile definition used by one or more user account definitions. Although a user definition does not have to use a group, doing so makes it much easier to maintain sets of similar user definitions.

Users User account definitions specify which users are allowed to log onto the FTP server and which files they may access.

For information about how group and user definitions are used, refer to “[Defining Users](#)” on page 74.

• Group Definition

When [Groups](#) are selected from the above menu list, the currently defined group names are presented. You may select one of those group names and edit it by pressing **[Enter]**, delete it by pressing **[Del]**, or you may create a new group definition by pressing **[Ins]**.

A screenshot of a form titled "Groups". The form contains the following fields:
Group Name: Customers
Home Directory: /ftp:s
Login Msg Text: general.login:s
Hide Hidden: NO
Below these fields is a table with three columns: VIRTUAL, PHYSICAL, and PERMISSIONS.

The fields in a group definition provide inheritable values for user accounts using the group.

Group Name Group names may contain upper and lowercase letters, numerics, spaces, punctuation and other special characters, except for the open and closing brackets ([]). When choosing a

group name, use a name that is short enough to be remembered easily, but long enough to describe the group. For instance:

```
Customers
Dealers
Customers, special
```

Home Directory The home directory is a path specification for the directory that the user starts in. The path specification for a home directory may include an account name.

Users connecting to the FTP server may be able to access directories subordinate to this home directory, but not directories or accounts higher up in the directory tree structure. For more information about home directories, refer to “[Controlling Access to Your File System](#)” on page 69.

Login Msg Text Specifies the file name with optional path for the text file displayed when the user successfully connects to the FTP server. When a full path is not specified, the file name is relative to the user’s home directory.

For more information about login message files, refer to “[Using Custom Messages](#)” on page 78.

Hide Hidden A “YES” specification suppresses directory listings of files and directories marked with the hidden attribute. These hidden files may still be accessed, but only if the user knows that they are there and requests them by name.

This feature can be useful for hiding the login and directory change message text files in directory listings. See “[Using Custom Messages](#)” on page 78 for more information about this topic.

The bottom portion of the group definition screen is devoted to virtual directory definitions. There must be at least one virtual directory defined for a user, either in this group definition, if the user definition uses a group, or in the user definition. For more information about using virtual directories, refer to “[Virtual Directories](#)” on page 71 and “[Using Groups](#)” on page 74.

Virtual Directories

You can edit an existing virtual directory definition by positioning the highlight bar on the definition and pressing **Enter**, delete a definition by positioning to it and pressing **Del**, or create a new definition by pressing **Ins**. When editing or creating a definition, the following screen is used:

Modify Dir	
Virtual	/
Physical	/ftp:s
Read access	YES
Write access	YES
Listing	YES
Create	YES
Delete	YES
Modify	YES
Subdirectory	YES

Virtual This is the directory name that is presented to the FTP client when they view the directory listing.

Physical This is the path and name of the directory, as it exists on this system. When the path is relative, that is, it does not start with the root directory, it is relative to the home directory.

The path may start with an account name specification.

Read access Allows the user to download the files. Specifically, the following FTP commands can be used on the files in the directory:

Server Command	THEOS FTP Client Command
MDTM	REMOTE MDTM
RETR	GET or RECV, VIEW
SIZE	REMOTE SIZE

Write access Allows the user to upload a file to the directory. Specifically, the following FTP commands can be used on the files in the directory:

Server Command	THEOS FTP Client Command
APPE	APPEND
STOR	PUT or SEND

Listing Allows the user to list the files in the directory. Specifically, the following FTP commands can be used on the files in the directory:

Server Command	THEOS FTP Client Command
LIST	DIR, LS, SAVEDIR and SAVELS

Create Allows the user to create new subdirectories in this directory. Specifically, the following FTP commands can be used on the files in the directory:

Server Command	THEOS FTP Client Command
MKD	MD or MKDIR

When they do create a new subdirectory, and the [Subdirectory](#) permission is set, the new subdirectory will have the same access permissions as this directory.

Delete Allows the user to delete this directory. Specifically, the following FTP commands can be used on the files in the directory:

Server Command	THEOS FTP Client Command
RMD	RD or RMDIR

Modify Allows the user to delete and rename files in the directory. Specifically, the following FTP commands can be used on the files in the directory:

Server Command	THEOS FTP Client Command
DELE	DELETE or REMOVE
RNFR	RENAME
RNTO	RENAME

Subdirectory Indicates that the access permissions apply to subdirectories of this directory.

To read more information about virtual directories and access permissions, refer to “[Controlling Access to Your File System](#)” on page 69.

• User Definition

When [Users](#) is selected from the menu list, the currently defined user account names are presented. You may select one of those user names and edit it by pressing **[Enter ↵]**, delete it by pressing **[Del]**, or you may create a new user definition by pressing **[Ins]**.

Users

User Name:	JaniceDoe	
Group Name:	Customers	
Home Directory:	/ftp:s	
Login Msg Text:	general.login:s	
Hide Hidden:	YES	
Password:	*****	

VIRTUAL	PHYSICAL	PERMISSIONS
/	/ftp:s	RLS
/incoming	/ftp/incoming:s	WLCS
/janice	/customer/sysgy:s	RWLCDS

Setup

The fields in a user definition provide the specific values used when a user logs onto the server with this as the user account.

If a [Group Name](#) is specified, then any of the fields [Home Directory](#), [Login Msg Text](#) and [Hide Hidden](#) may be left blank and the values from the group definition are used when a user connects and specifies this user account. In addition, if there are virtual directories defined in the group definition, those directory definitions are merged with any virtual directory definitions defined for this user. A user-account-defined virtual directory name will replace an identically named group-defined virtual directory definition. For more information, refer to “[Using Groups](#)” on page 74.

User Name The name defined here is the account name that a user can log onto when they connect to the FTP server. A user name may be long and may contain upper and lowercase letters, numerics, spaces, punctuation and other special characters except for the open and closing brackets ([]). User names are maintained as typed, but they are case-insensitive when a user attempts to log on. When choosing a user name, use a name that is short enough to be remembered easily, but long enough to identify the user.

You may define a special user named “anonymous.” This is the special account definition used by people connecting as ANONYMOUS. It must be defined to allow anonymous connections (see “[Anonymous Users](#)” on page 84). Leave the fields [Group Name](#) and [Password](#) fields blank when defining an anonymous user.

Even if specified, they are ignored when a user connects as ANONYMOUS.

Group Name Enter the name of a group definition. Group names are case-insensitive. This field may be left blank if you do not want this user to inherit the properties defined for a group.

When a group name is specified, the group does not have to be defined at this time.

Home Directory The home directory is a path specification, including an optional account name, for the directory that the user starts in.

Users connecting to the FTP server may be able to access directories subordinate to this home directory, but not directories or accounts higher up in the directory tree structure. For more information about home directories, refer to “[Controlling Access to Your File System](#)” on page 69.

Login Msg Text Specifies the file name with optional path for the text file displayed when the user successfully connects to the FTP server. When a full path is not specified, the file name is relative to the user’s home directory.


For more information about login message files, refer to “[Using Custom Messages](#)” on page 78.

Hide Hidden A “YES” specification suppresses directory listings of files and directories marked with the hidden attribute. These hidden files may still be accessed, but only if the user knows that they are there and requests them by name.




This feature can be useful for hiding the login and directory change message text files in directory listings. See “[Using Custom Messages](#)” on page 78 for more information about this topic.

Password Enter the password that must be used to log onto this account. Passwords are case-sensitive. You will be able to see what you type as you enter the password. However, when you move to another field, the password is displayed as a string of asterisks.

To specify that a user does not need a password when logging on, make sure that the password field is blank.

Each time that this field is selected (positioned to and **Enter** , is pressed), any current password is cleared.


The bottom portion of the user definition screen is devoted to virtual directory definitions. There must be at least one virtual directory defined for a user, either in the group definition, if the user definition uses a group, or in this user definition. For more information about using virtual directories, refer to “[Virtual Directories](#)” on page 71 and “[Using Groups](#)” on page 74.

You can edit an existing virtual directory definition by positioning the highlight bar on the definition and pressing **Enter** , delete a definition by positioning to it and pressing **Del** , or create a new definition by pressing **Ins** .

Refer to page 39 for a description of the fields used to define a virtual directory.

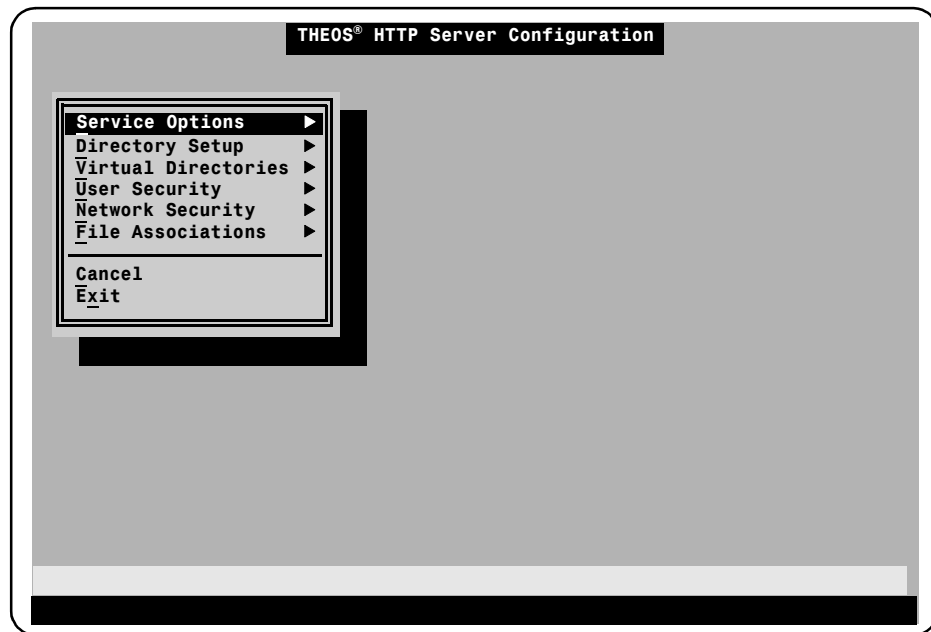
To read more about virtual directories and access permissions, refer to “[Controlling Access to Your File System](#)” on page 69.

■ Exit

This menu item exits Setup. Any changes that you made during this session must be saved in each of the configuration screens by using the **F10**  key to exit the individual screens.

Setup HTTP

The “THEOS HTTP Server Configuration” menu configures and displays the settings for the HTTP server.



Use the normal menu selection keys to select the desired item.

Service Options. Configures the global settings for the HTTP server (see page 45).

Directory Setup. Configures the directories used by the HTTP server for the log file, CGI applications and general resources (see page 48).

Virtual Directories. Configures the virtual directories used by the HTTP server and their relationship to the real directories on your system (see page 49).

User Security. Configures the user security associated with specific virtual directories (see page 50).

Network Security. Configures the lists of IP addresses and domain names that are allowed to connect to or are denied access to this HTTP server (see page 51).

File Associations. Configures the association between file extensions (file types) and the application used to view or process the file (see page 52)

Cancel. Menu item used to exit without saving any of the changes made during this Setup session.

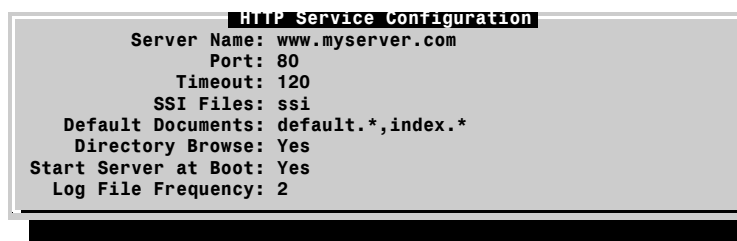
Exit. Menu item used to save all changes made and exit Setup.

When making changes or additions to the information in the various screens in this setup, it is best to use the **[F10]** key to save the change and return to the next higher level window. Using the **[Esc]** key will cancel the last change or addition.

Changes to the HTTP configuration settings do not take effect until the next time that the server is started. If it is currently running, it must be stopped and then restarted for these new settings to take effect.

■ Service Options

Most of the basic parameters used by the HTTP Server are defined with this menu item.



Server Name Specify the name of the server such as “www.theos-software.com” or “www.myserver.com.” If you do not have a registered domain name, you may use the IP address of your machine, such as “192.168.100.1.”

Port This is the port number that the server listens to for incoming requests. The default port number is 80 and, in most situations, should be the value used.

You may use any port number that is unused by other servers on your system. However, when a port number other than 80 is used, you will have to tell all users who want to access your server what that port number is. HTTP user agents will use port number 80 unless they are specially configured.

A list of the well-known services and port numbers used by those services can be found in the file `SYSTEM.TEOS32.SERVICES`.

Timeout The timeout value defines the number of seconds that the server keeps a connection open without a response from the client. Many HTTP clients open multiple connections to the server (a separate connection for each component of the document, such as an image or sound file). Each connection remains open for the amount of time specified here.

The timeout value should not be too small or the connection is closed before the transaction is complete and must be reopened causing additional overhead for the transmission. When the timeout value is too large the connection remains open too long and might cause your system to run out of task space and numbers.

The default for this value is 120 seconds.

SSI Files This field defines the list of file-types or extensions of documents that are preprocessed by the server before giving the document to the client. More than one file-type can be specified by listing the file-types, one after the other, separated by a single comma character.

The default file-types that are preprocessed by the server are SHTML.

Any document containing SSI directives or ASP statements that does not use one of the file-types specified in this setting is not preprocessed and the directive/statement is ignored by the client (SSI directives and ASP statements are defined within HTML comment delimiters).

Default Documents This field defines the default document names used when a non-specific request is made. For instance, when this field specifies "index.html" and a request is made for "http://server-name/" the server will look for a file with a name of index.html in the virtual root directory.

Multiple document names may be specified by listing the names, one after the other, separated by a single comma character.

The file names for the default documents may be specified with wild cards including the "*", "#", "@" and "?" specifiers. For instance, the default for this field is: "default.*."

Directory Browse Enables or disables directory browsing. When enabled and when a client requests a directory that does not contain one of the default documents (see “[Default Documents](#)” above), a hyperlink listing of the directory is sent to the client. When disabled and when none of the default documents exist, the client is informed that they do not have access to the site.

Refer to “[Directory Browsing](#)” on page 101 for a description of this feature.

Start Server at Boot A “YES” value starts the HTTP server each time that this system is booted. A “NO” value will not start the server automatically.

You can manually start and stop the HTTP server with the NET START command:

```
>net start http
```

```
>net stop http
```

Log File Frequency The code in this field controls whether the HTTP log file is created and the frequency that it is “rotated.”

- 0 Do not create or rotate the HTTP log file.
- 1 Create an HTTP log file but do not rotate it. Log file name is always HTTP.LOG.
- 2 Create an HTTP log file and rotate the log file daily. Log file name is HTyymmdd.LOG where *yy* is the year number, *mm* is the month number and *dd* is the day of the month number.
- 3 Create an HTTP log file and rotate the log file weekly, on Mondays. Log file name is HTyymmdd.LOG where *yy* is the year number, *mm* is the month number and *dd* is the day of the month number for the Monday of this week.
- 4 Create an HTTP log file and rotate the log file on the 1st of each month. Log file name is HTyymm01.LOG where *yy* is the year number and *mm* is the month number.

The location of the log file is defined by the [Logfile](#) field of the [Directory Setup](#) menu described next.

When the log [Log File Frequency](#) code is 2, 3 or 4, the log file is rotated at the indicated frequency. The actual rotation occurs when a new transaction is to be created in the log. If the period has elapsed and a new file has not been created yet, the current file is closed and a new file is opened.

For instance, the frequency code is 3 and the last activity was on Wednesday, May 27, 1998. The current log file is named HTTP_LOG.19980525 which is the date for the Monday of that week. If a new entry is to be added on Thursday, June 4, 1998 the current file is closed and a new file is started named HTTP_LOG.19980601, which is the date for the Monday of that week.

■ Directory Setup

Several directories are used by the HTTP Server for specific purposes. This screen allows you to define the location of those special directories.



The screenshot shows a window titled "HTTP Directories" with a light gray background. It contains three lines of text: "Logfile: /logs:s", "CGI: /cgi:s", and "Resources: /httpicon:s". The window has a thin black border and is set against a dark background.

Logfile Defines the directory used for HTTP log files.

The directory specified here is not created automatically. If the directory does not exist then no logging is performed. When it does exist when the HTTP server is started, the appropriate log file is created or appended to if it already exists. Refer to the [Log File Frequency](#) field in the [Service Options](#) menu for information about the names used for HTTP log files.

Refer to Appendix D: "[HTTP Log File](#)" starting on page [133](#) for a description of the format of records created in the HTTP log file.

CGI Defines the directory used by CGI applications. Only CGI applications in this directory or in its subdirectories can be accessed and used by web documents on this server.

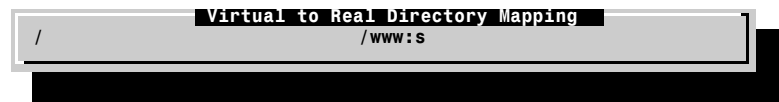
The directory specified is used for execute-only CGI scripts.

Resources Defines the directory used for common resources such as web document error messages, image files for directory listing, *etc.*

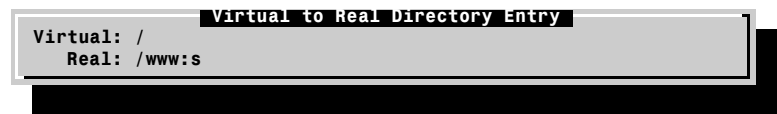
The default for this directory is /httpicon which is the directory supplied during THEO+Server installation. It contains the image files used for directory browse displays and the active server pages displayed for server error message documents.

■ Virtual Directories

The HTTP Server does not provide clients with access to the real directories on your system. Instead, it provides clients with access to virtual directories. The relationship of these virtual directories to the real directories on your system is defined in this screen.



You can add a new virtual directory with the **Ins** key or change an existing definition by positioning to that definition and pressing **Enter**. An existing definition can be removed by positioning to that definition and pressing the **Del** key.



Virtual Specify the name of the virtual directory.

Real Specify the path to the real directory for the virtual directory specified above. This specification is a standard THEOS path specification and may include an account name and drive code.

A virtual root directory must be defined. The recommended mapping for this directory is to the /www:s directory created when THEO+Server is installed.

Defining a virtual directory name for a real directory that contains subdirectories provides access to all of those subdirectories in the real directory.

Do not define a virtual directory name of “/cgi” because this is a reserved virtual directory name. You may define different virtual directory names that map to the real directory containing your CGI applications.

■ User Security

The user security screen allows you to specify protection for a virtual directory so that unauthorized users cannot access the documents in that directory or its subdirectories.

Refer to “[Controlling Access to Specific Directories](#)” on page 96 for a description of user security and its operation.

Password Protected Directories		User names
/Examples	Sample files and programs	Joseph
/Private/Stuff	Personal files and documents	Douglas

Selected Path and Area
Path: /Examples Area name: Sample files and programs

Select a directory displayed in the top-left window pane and press **Enter** to make changes to its definition. You may also press **Del** to delete the definition or **Ins** to define a new protected directory.

Path Enter the virtual directory path that you want protected.

Area name Enter a description of this virtual path. The name entered here is given to the client when asking for the user name and password to access this directory.

Use an area name that is informative enough so the user will know what area they are trying to access and which name and password they should use.

To add, change or delete the users that are allowed to access the directory, press the **Tab** key or the **→** key to position to the top-right window pane and then position to the desired name and press **Enter** or **Del** to change or delete it. Press **Ins** to add a new user that can access the directory.

Selected User Name
User name: John Doe Password: Open99Sesame88Please

The user name and password specified here is the information that must be supplied to access any of the documents in the virtual directory specified above or in any of its subdirectories.

User name Enter a user name or other term that must be entered to gain access to the protected directory. This name is case-insensitive.

You may enter a mixed case name and the user may respond with a name in all uppercase and it will match.

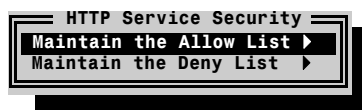
Password Enter the password for this user name. The password is case-sensitive and the user must match the password specified here exactly or they will be refused access.

Before pressing **[Enter ↵]** or moving to a different field or window, examine the password that you have entered. This is the last time that you will see this password in plain text. When changing a user definition, the existing password is displayed as eight asterisks (*********) and it is encoded in the authorization file.

The length allowed for the user name and password fields is limited to 127 characters for a combination of both fields.

■ Network Security

This menu allows you to define and maintain lists of IP address or names that are allowed or denied access to this HTTP server. Any changes to these settings will not take effect until the next time that the server is started.



The allow and deny lists are lists of IP addresses that the HTTP Server uses to determine whether or not a remote site is allowed to connect to the server. The allow list is a list of addresses, address ranges and address subnets that are specifically allowed to connect to the server. The deny list is a similar type of list of addresses that are specifically not allowed to connect to the server.

Refer to “[Allow IP and Deny IP Lists Limit Access to the Server](#)” in Chapter 4 “[Server Security](#)” starting on page 55 for a description of the operation of these IP lists.

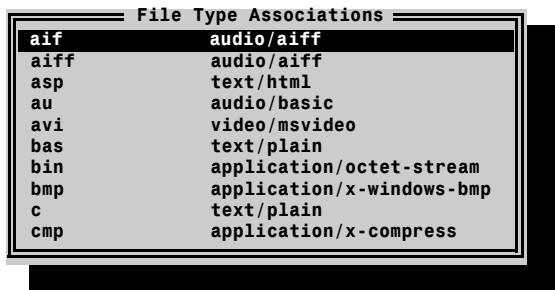
Select the list that you want to view or maintain and press **[Enter ↵]**. A window appears showing the existing IP addresses defined for that list.

As the instructions at the bottom of the screen indicate, use the **[Ins]** key to add a new entry, the **[Del]** key to deleted a selected entry, the **[F10]** key to save the changes and exit to the main menu, and the **[Esc]** key to exit to the main menu without saving any changes.

Remember, changes to these settings will not take effect until the next time that the server is started. If it is currently running, it must be stopped and then restarted for these new settings to take effect.

■ File Associations

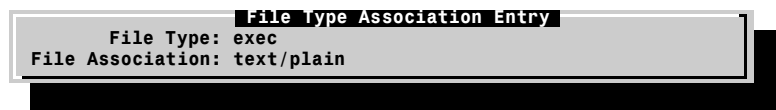
This screen defines the relationships between a file's file-type or extension and the MIME file type specification.



aif	audio/aiff
aiff	audio/aiff
asp	text/html
au	audio/basic
avi	video/msvideo
bas	text/plain
bin	application/octet-stream
bmp	application/x-windows-bmp
c	text/plain
cmp	application/x-compress

Every time that a request is made for a file or document, the server must create MIME headers for the document before it delivers the document or file to the client. These MIME headers tell the user agent the type and format of the information in the file and what it should do with it.

Generally, the file type associations provided with the default configuration are adequate. You can add your own association with the **[Ins]** key or change an existing association by positioning to that association and pressing **[Enter ↵]**. An existing association can be removed by positioning to that association and pressing the **[Del]** key.



File Type:	exec
File Association:	text/plain

File Type This is the file-type as specified on the THEOS system.

File Association: The MIME header for “Content-Type:” that should be used when delivering a file with the file-type indicated.

The meaning of most of the Content-Type MIME headers is described in RFC 2045-2049 (Multipurpose Internet Mail Extensions).

■ Cancel

This menu item will exit the HTTP configuration **without** saving the changes that you may have made during this setup session.

■ Exit

Any changes made during this setup session are saved in the SYSTEM.THEOS32.HTTPDCFG file and setup returns to the SETUP SERVER menu.

4 Server Security

A server on a network can be a significant security issue whether the server is accessed by LAN, WAN or Internet users. The problem is that unknown users might access the server system and potentially harm the integrity of the database on the server's system. There are two basic mechanisms provided by THEO+Server to reduce the risk involved with network servers.

You can try to control who has access to the server. This is done with the allow/deny IP address lists identical to but separate from the configuration used by THEO+Net.

You can also control access to a server system's files by users that are allowed to connect to the server.

Finally, you can log user activity on a server by users who do connect to the server.

■ Controlling Access to the Servers

If your system has connections to insecure networks, such as the Internet, you should consider defining IP addresses or subnet address that you want to have access to your FTP Server and to your HTTP Server. Restricting or allowing specific IP addresses is controlled separately for each server on your system.

Whether or not your system is connected to the Internet, you may want to limit your FTP server so that only certain sites on the network are allowed to access the server. For instance, the local network may have sites located in the administration office and other sites in the production area and you only want the sites in the office to be able to use this server for file transfer purposes.

The FTP Server restricts access to users by using user accounts. If you do not tell a person the user account name and its password, remote users cannot connect to the server. See "[Controlling Access to the Server's File System](#)" on page 59 for information about defining user accounts. The individual user accounts limits which files the user is allowed to download and where they can upload files.

Remote users on the Internet may try to connect to your server if they know the server's IP address on the Internet. If you have a permanent connection to the Internet, your IP address is constant and remote users may be able to find your address if you have registered it with InterNIC or if

you have published the address in a location where they can find it. For instance, your ISP may offer a directory of users or you may have a web page on their system that lists your IP address for connections.

If your connection to the Internet uses dynamically assigned IP addresses, which are used by most dial-up connections, then your IP address may be different each time that your system connects to the Internet. If your Internet access is provided by a proxy server on a Windows 95 or Windows NT system, there are utilities available that can determine your current IP address and automatically publish that information in a static location, such as a web page. Some are freeware and others shareware. Use your Windows system web browser to view and search the site:

<http://www.windows95.com/apps/netapps.html>

Or use one of the search engines available on the Web and search for sites containing the key words “dynamic IP.”

When trying to determine whether or not a remote Internet user will be allowed access to your system, you must consider whether or not they have a static IP address or if they will be using dynamically assigned IP addresses. If their address is static, merely add it to the allow list of IP addresses.

If their address is dynamically assigned, you will have to use a subnet specification for their normal ISP assignment. Have them contact their ISP to determine the range of possible addresses that may be assigned when they connect to the Internet. Using this information, define the IP address subnet(s) or subnet ranges that will include all of the possible addresses that they might be assigned.

Although you should always use network user names and passwords, when the remote user has dynamically assigned IP addresses, network user names and passwords are particularly important. You want to prevent other customers of that ISP from gaining access to your system.

If your Internet access is provided by a PPP connection and the DialNet command, it displays and logs your IP address each time that it connects to your ISP. Refer to the description of that command in the *THEO+Net Installation and Reference* manual.

■ Allow IP and Deny IP Lists Limit Access to the Server

The allow and deny lists are lists of IP addresses that a server uses to determine whether or not a remote site is allowed to connect to the server. The allow list is a list of addresses, address ranges and address subnets that are specifically allowed to connect to the server. The deny list is a similar type of list of addresses that are specifically not allowed to connect to the server.

Both lists of IP addresses are used in sequence. For instance, if a site matches one of the entries in the allow list and also matches one of the entries in the deny list, the site is not allowed to connect.

A list is only checked if there is at least one entry defined in the list. For instance, if there are no entries in the allow list but there is an entry in the deny list, then all sites are allowed to connect except those listed in the deny list.

Additional information about the usage of allow and deny lists can be found in the chapter “Network Security” in the *THEO+Net Installation and Reference* manual. Although the sets of lists (one for the FTP server, one for the HTTP server, another for the Login server, *etc.*) are separate, they all operate in the same manner.

As initially installed, the FTP server and the HTTP server have no IP access restrictions. Thus, all sites may access the server and none are denied.

An entry in the allow or deny list may be specified in one of several forms:

- ▶ The host name as defined in the file `SYSTEM.THEOS32.HOSTS`. This file is maintained by the Host Names Database screen in the Setup NET command.

my-company

- ▶ The domain name as defined by the Domain Name Service specified in DNS Servers screen in the Setup NET command.

myisp.com

- ▶ The dotted IP address for a site.

207.21.75.100

- ▶ The dotted subnet IP address for a group of sites.

192.168.*.*

With this syntax, all sites whose IP address matches the wild-card

specification are treated as a match for the list.

- A range of IP addresses or subnet IP address.

192.168.50.1-48

This syntax specifies that all sites will match if their IP address is 192.168.50.1, 192.168.50.2, ..., 192.168.50.48.

The CIDR form may also be used for specifying a range of IP addresses:

192.168.50.1/48

Generally, specific IP addresses for individual nodes or subnet addresses for groups of nodes that you want to have access to your system are defined in the allow list. When subnet addresses are defined in the allow list, specific IP addresses of nodes on those subnets that you do not want to have access to your system are listed in the deny list. For instance:

Allow List:

192.168.38.*
207.21.75.*

Deny List:

207.21.75.5
207.21.75.7

These specifications state that everyone on the local network identified by 192.168.38.* is allowed to connect to the server. Also, everyone in the subnet 207.21.75.* is allowed to connect except the two nodes 207.21.75.5 and 207.21.75.7, which are specifically denied access to this Server.

■ Controlling Access to the Server's File System

Similar to the way that the Network Login Server allows you restrict who can connect as a user to the Login Server with network user names and passwords, user accounts are also used by the FTP Server and, to a small extent, the HTTP Server.

An FTP **user account** is a name and password that a remote user must supply to gain access to the FTP Server. In addition, the FTP Server allows you to define a special user account named “Anonymous” that a remote user can use to connect to your FTP Server. Each FTP user account, including the anonymous user account, specifies a home directory for the account and a set of virtual directories. The remote user can only access files in these directories or in directories subordinate to these directories.

The HTTP Server does not define separate user accounts but instead has a single, implied anonymous user account. All remote users connecting to the HTTP Server use this implied anonymous user account. Like the FTP Server, the HTTP Server's anonymous account specifies a home directory and a set of virtual directories. The remote user can only access files in these directories or in directories subordinate to these directories.

The **home directory** for a user account is the directory used as the virtual root directory by a user. A remote user cannot access directories higher in the virtual directory tree than the home directory. A home directory is also the home or initial directory accessed. For instance, an anonymous FTP user account could be defined with a home directory of `/ftp/incoming:s`. Any user logging in as anonymous would have initial access to the `/ftp/incoming` directory (this would be shown as the root directory when the remote client performs a `dir` listing). They could access directories that are subordinate to this directory but would have no access to any files in directories higher-up in the directory tree or to any “sibling directories.”

For instance, if the real directory tree on the server was:

```

/
├── ftp
│   ├── files
│   └── incoming
│       └── files
├── programs
├── www
│   ├── examples
│   └── pages

```

When the home directory is defined as `/ftp/incoming`, the user could access that directory and the `/ftp/incoming/files` directory (because it is subordinate to `/ftp/incoming`) but would have no access to `/`, `/ftp`, `/ftp/files`,

/programs, or /www directories because they are either higher-up the directory structure or are sibling directories to /ftp/incoming.

■ Virtual Directories Limit Access to Directories

Neither the FTP Server nor the HTTP Server allow a remote user to access the real file system on the server's computer. Both use the concept of virtual directories. A **virtual directory** is a fictitious name that is defined in a server's configuration. This virtual directory name need not be the name of a directory that actually exists on the computer. Instead, it is a name that is mapped to an existing, real directory name on the server computer's file system.

In the previous example, the home directory is a special virtual directory specifying that the virtual root is mapped to /ftp/incoming:s. An HTTP server might map the virtual root to /www:s. When a user connects to the server they only see the virtual directory names. In this example, they will see "/" as the path name even though they are actually using the real path of "/ftp/incoming:s" or "/www:s." If additional, subordinate virtual directories were defined, such as "/ftp/sample" mapped to "private\programs:a," the user would only see "/ftp/sample" even though they were accessing the real directory of "private\programs:a."

In addition to the virtual root directory, both servers allow you to define a complete virtual directory tree. The FTP Server allows you to define a different virtual directory tree for each user account and a virtual directory tree for the anonymous user account. The HTTP Server, having only an implied anonymous user account, can only define a single virtual directory tree.

With virtual directories, you define the directories that a user has access to and the relationship between these virtual directories. The FTP Server also allows you to specify what type of access the user is allowed for each of the directories defined in the virtual directory tree. The HTTP Server doesn't provide or need this capability because it only reads files and never tries to execute, write or delete them. (The CGI directory used by the HTTP Server is a special directory defined outside of the virtual directory tree. Files in the CGI directory are executed but not read by the server.)

Refer to the instructions specific to setting up each server for information about defining a virtual directory structure. These can be found in "[Virtual Directories](#)" on page 39 of Chapter 3 "[Setup Command](#)" for the FTP server and in page 49 for the HTTP server.

A discussion of how virtual directories are defined and used in the FTP Server can be found in "[Controlling Access to Your File System](#)" on page 69. The

HTTP Server virtual directories are discussed in [“Controlling Access to Your File System”](#) on page 94.

■ **Logging User Activities**

Both the FTP server and the HTTP server can create a log of user activity on the server. These logs are useful for determining who is using the server and what kind of activities they are performing.

■ **FTP Server Logging**

With the THEOS FTP server you can keep a log of user activity on the server. There is a wide range of activities and events that you can log:

- ▶ When the server was started and stopped
- ▶ Who attempted to or did log onto the server
- ▶ What commands the user used
- ▶ What replies to those commands were produced
- ▶ What files were uploaded by the user
- ▶ What files were downloaded to the user

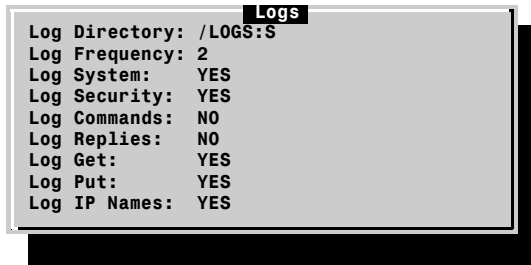
To enable this logging, the Setup Server, FTP, [“Log Options”](#) menu is used.



Logging is enabled when a directory name is specified in the “Log Directory” field and that directory exists when the server is started, the “Log Frequency” is not zero and one or more of the other fields is enabled. When “Log Directory” is blank or “Log Frequency” is zero no logging is performed.

When first setting up the server or when a user reports problems connecting to or using your server, you may want to enable all of the log options. Enabling all of the options causes almost everything that happens with and to the user to be written to the log file.

For normal operations, the recommended settings are:



The above settings log all of the major events that happen to your server. With this information, you could easily write an application program in MultiUser BASIC that analyzes the file and reports on how many times a file was downloaded, what types of users (domain names) are accessing the site, etc. The layout and definition of the log file entries are described in Appendix C: “FTP Log File” on page 131.

When logging is enabled and the server is running, the log file is kept open by the server. This means that it cannot be deleted until the server is stopped. However, you can still access and list the file or make a copy of it while the server is running. If you want to track the events that are being logged to the file, you can use the TAIL utility command to view the records as they are being added to the file:

```
>tail /logs/ft980704.log (follow
```

Log files are never deleted automatically and, if the log frequency is set to 1, it may grow to be a very large file. The file or files should be periodically erased or moved to an archive location to prevent a disk-full condition.

■ HTTP Server Logging

With the THEOS HTTP server, you can keep a log of user requests to the server. There are two types of requests that are logged:

- ▶ GET requests which log every document or file retrieval by the browser.
- ▶ POST requests which log every POST of a FORM from the browser.

To enable this logging, the Setup Server, HTTP, “[Directory Setup](#)” menu is used.



Logging may be enabled when a directory name is specified in the “Logfile” field and that directory exists when the HTTP server is started. When “Logfile” is blank or when the directory does not exist, no logging is performed. To actually enable logging when there is a directory specified in this field, you must also set the [Log File Frequency](#) field in the [Service Options](#) menu to a nonzero value (see page 47).

The information logged with each event includes:

- ▶ IP address of requesting user
- ▶ Server name
- ▶ Account name if available
- ▶ Date and time of request
- ▶ Request type
- ▶ Request result code
- ▶ Document URI
- ▶ Browser identify

The specific layout and definition of the log file entries are described in Appendix D: “[HTTP Log File](#)” on page 133.

When logging is enabled and the server is running, the current log file is kept open by the server. This means that it cannot be deleted until the server is stopped. However, you can still access and list the file or make a copy of it while the server is running. If you want to track the events that are being logged to the file, you can use the TAIL utility command to view the records as they are being added to the file:

```
>tail /logs/ht980704.log (follow
```

Log files are never deleted automatically and, if the log frequency is set to 1, it may grow to be a very large file. The file or files should be periodically erased or moved to an archive location to prevent a disk-full condition.

Part II ---

THEOS FTP Server

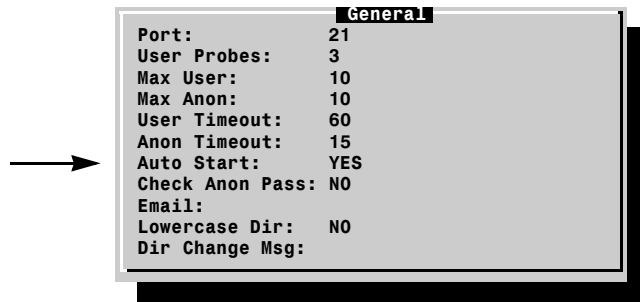
5 Using the FTP Server

■ Starting and Stopping the Server

There are two ways to start the THEOS FTP server: automatically and manually.

■ Starting the FTP Server Automatically

To start the FTP server every time that this system is booted, enable the “Auto Start” item in the Setup FTPD, “[General Options](#)” menu.



Use the **[F10]** key to save your change. The next time that this system is booted, the FTP server is started automatically.

■ Starting and Stopping the FTP Server Manually

To start the FTP server manually, use the NET START command:

```
>net start ftpserver
Task started as process #61.
```

The FTP server is stopped with the NET STOP command:

```
>net stop ftpserver
```

As a convenience, the FTP server can be started and stopped with the name “FTP:”

```
>net start ftp
```

```
>net stop ftp
```

■ Using the FTP Server with Dial-Up Networking

If your FTP server system accesses the Internet with a dial-up networking connection, that connection must be made by you before anyone can access your server. This is done with the DialNet command described in the *THEO+Net Installation and Reference* manual. For instance:

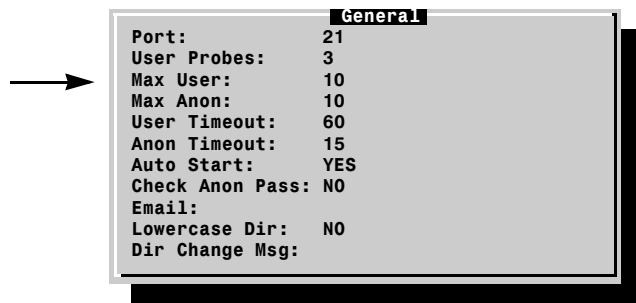
```
>dialnet start myisp
Profile "MYISP" connected successfully,
Host IP address: 206.163.58.53
Remote IP address: 204.245.231.131.
```

■ Limiting the Number Of Connections

When the FTP server is installed on a system that is also used for other purposes, you may find it desirable to limit the number of users that may connect to the FTP server at any one time. Each user connection to the server:

- ▶ Requires a user partition. Every system has a maximum number of user partitions that it supports. This limit is defined in the system configuration maintained with the SysGen utility, “Maximum Number of Tasks” field.
- ▶ Uses a lot of system resources. Most users on a system (terminals and workstations) actually make few demands on a system because they are mainly used for data entry and lookups. However, an FTP client connection may perform almost continuous disk access and data transmission with little time spent waiting for operator input.

To limit the number of FTP clients that may connect to your server at any one time, specify the limit in the “Max User” item in the Setup FTPD, “[General Options](#)” menu.



You may also want to limit the length of time that an idle user may remain connected, particularly when the number of connections allowed is low. The “User Timeout” field specifies the number of minutes that an idle user will remain connected. When the user remains idle for this length of time, they are automatically disconnected allowing another user to connect to your server.

There are separate limitations that you can impose upon guest or anonymous user connections. You may wish to limit the number of guest logins so that more of the allowed users are reserved for your customers. The fields “Max Anon” and “Anon Timeout” apply only to guest logins. These values apply to guest logins and are only used if they are lower than the corresponding limits for nonguest accounts.

■ **Controlling Access to Your File System**

If you wanted to, you could configure your FTP server to allow all users to access all of the files on your system. This is done by defining an anonymous user whose home directory is your system’s root directory and with a virtual directory that maps to the root directory for each of the drives on your system with all access permissions enabled. For instance:

Users		
User Name: Anonymous		
Group Name:		
Home Directory: /:s		
Login Msg Text:		
Hide Hidden: NO		
Password:		
VIRTUAL	PHYSICAL	PERMISSIONS
/	/:s	RWLCDMS
/a	/:a	RWLCDMS
/b	/:b	RWLCDMS

When a user connects to your FTP server as an anonymous user, they would have full access to the files owned by the SYSTEM account on the S, A and B drives. In almost all cases, this is not a desirable situation.

Generally, you want to provide a user with access to only certain directories and files on your system, and you want anonymous users to have even less access to your system (see “[Anonymous Users](#)” on page 84). With the THEOS FTP server, restricted access is provided in two ways: by limiting which directories they can access and by controlling the types of access that they will have to the directories and the files in the directories.

■ Home Directories

The home directory defines the virtual root and the initial directory seen by a user. You should not specify the real root directory of a drive on your system for this home directory. If you do, the user may have access to all of the directories and files on that drive.

To provide the most security, create a special directory that is used just for this home directory field. For instance:

```
>mkdir /ftp:s
```

It is not necessary to have any files in this special directory. The virtual directory definition for each use account can make it appear that all of the directories and files that you want the user to see are located in this home directory.

The home directory does not have to be owned by the SYSTEM account nor does it have to reside on the s drive. The specification for the home directory can include an account name specification and a drive code. For instance, you could define the home directory as:

```
private\special\files\ftp:c
```

When a user connects to the server, the home directory is displayed as and accessed by the root directory reference “/”. Once connected, a user cannot access or refer to directories above this home directory. For instance, when the home directory is the current working directory, they cannot change to “..” even though they might know that the home directory is not the real root directory of the server.

Because the home directory is part of a user account definition, each user can have a different home directory. For instance, the home directory for the anonymous user might be “/ftp/anon:s” (assuming that that is a real directory on the system). Other users might have home directories of “/ftp:s.” An anonymous user would not have access to the files or directories in “/ftp” except for those directories and files subordinate to “/ftp/anon:s.”

■ Virtual Directories

A virtual directory defines the relationship between what the user sees as directories on your system and what the actual directories are. For instance, the real directory structure on your system might be:

System account:

```

{fax
└─example
/ftp
/ftp.files
└─incoming
/ftp.messages
/include
{theomail
└─alice
└─george

```

Develop account:

```

/includes
/project1
└─doc
└─helps
└─messages
└─source
/project4
└─doc
└─helps
└─messages
└─source

```

But, when the programming engineer responsible for working on the “Project4” product connects to the server, you might want her to only see the following directory structure:

```

/
└─includes
└─incoming
└─project1
└─└─helps
└─└─messages
└─└─source
└─project4
└─└─doc
└─└─helps
└─└─messages
└─└─source

```

With virtual directory definitions, this apparent directory structure can be easily defined, even though it groups together directories from different accounts.

Users		
User Name: Tawnya-4		
Group Name:		
Home Directory: /ftp:s		
Login Msg Text:		
Hide Hidden: NO		
Password: *****		
VIRTUAL	PHYSICAL	PERMISSIONS
/	/ftp:s	RWLCDMS
/incoming	/ftp.files/incoming:s	WS
/includes	develop\includes:s	RWLCDMS
/project1	develop\project1:s	RLS
/project4	develop\project4:s	RWLCDMS

Remember, if a virtual directory is not defined, the user has no access to the directory, even if that directory is the home directory for the user.

Always define the virtual directory root to be the same as the home directory.

■ Access Permissions

The virtual directories described in the prior section define a name and a path to a real directory on your system. This virtual directory definition gives the user access to the directory but it doesn't define what type of access the user has.

The access permissions of a virtual directory definition give you the capability of controlling the type of access that a user has to the directory and files in a directory. With access permissions, you control whether the user can:

- ▶ Download files from the directory
- ▶ Upload files to the directory
- ▶ Delete files in the directory
- ▶ Create new subdirectories of the directory
- ▶ Delete a subdirectory of the directory
- ▶ View the directory listing of files in the directory

If you do not specify that a user has a specific type of access to a directory, then they do not have that access. For instance, notice the access permissions for the example shown on page 71.

VIRTUAL	PHYSICAL	PERMISSIONS
/	/ftp:s	RWLCDMS
/incoming	/ftp.files/incoming:s	WS
/includes	develop\includes:s	RWLCDMS
/project1	develop\project1:s	RLS
/project4	develop\project4:s	RWLCDMS

In the second virtual directory definition for the “/incoming” directory, the access permissions are “WS.” This states that the user may write or upload files to the directory (W) and that these permissions apply to subdirectories of this directory (S). Since no other permissions are given, the user cannot download files, create new subdirectories, delete subdirectories, delete files in this directory or its subdirectories. In fact, since no permission is given for listing the directory contents, the user cannot do that either.

The definition for the “/project1” directory states that the user can download files from the directory (R), list the directory contents (L), and that these two permissions apply to any subdirectories of the directory (S). Again, since no other permissions are specified, that is all that the user may do with this directory.

■ **Defining Users**

As mentioned in Chapter 4 “[Server Security](#)” the FTP Server requires remote users to specify a user name and password when they connect. Each user account defines the specific virtual directories that the user may access and the access permissions for those directories.

You could define a single user name used by all users which might be appropriate for some servers. For most servers it is more appropriate to define a user account for each user that you want to access your system. Defining a user account for each user allowed access to your system increases the security of the system by allowing you to define the specific access that each user may have and it also allows you to identify each user in the server log file that can be generated.

■ **Using Groups**

A server that has dozens or maybe even hundreds of different user accounts can be awkward to create and maintain. Most systems with numerous user accounts generally have just a few different types of accounts. For instance, a system might have the following types of user accounts defined:

- ▶ One or two users that need full access to the system for general maintenance purposes.
- ▶ Several users that need to be able to contribute or upload files to specific directories on the server.
- ▶ Numerous users that only download files from the server with occasional contributions.

Group definitions make this type of system much easier to maintain user account definitions. A group definition is created for each of these three basic types of users and it defines the common attributes for each type of user. After the common attributes are defined for each type or group of users, the user accounts are created specifying that they are a member of one of the groups. Only the differences between the common attributes and the specific attributes needed by the individual user account has to be specified for each user account definition. Frequently, this will be just a few fields.

In the above example, the three groups could be defined as:

Group name:	Maintain	
Home directory:	/ftp:s	
Login Msg Text:	/ftp.messages/maintain.login:s	
Hide Hidden:	NO	
VIRTUAL	PHYSICAL	PERMISSIONS
/	/ftp:s	RWLCDMS
/incoming	/new.files:s	RWLCDMS
/updates	/updates:s	RWLCDMS
Group name:	Contributor	
Home directory:	/ftp:s	
Login Msg Text:	/ftp.messages/contrib.login:s	
Hide Hidden:	YES	
VIRTUAL	PHYSICAL	PERMISSIONS
/	/ftp:s	RWLS
/incoming	/new.files:s	RWLCMS
/updates	/updates:s	RWLDS
Group name:	Customer	
Home directory:	/ftp:s	
Login Msg Text:	/ftp.messages/customer.login:s	
Hide Hidden:	YES	
VIRTUAL	PHYSICAL	PERMISSIONS
/	/ftp:s	RLS
/incoming	/new.files:s	WS
/updates	/updates:s	RLS

After these groups are defined, it is easy to create user accounts that use these group definitions. For instance, to create a new customer user account:

User name:	John Doe	
Group name:	Customer	
Home directory:		
Login Msg Text:		
Hide Hidden:		
Password:	special-words	
VIRTUAL	PHYSICAL	PERMISSIONS

Only the fields “User name,” “Group name” and “Password” are filled in. The fields with no entry in them will inherit the definition for those fields from the current definition for the group “Customer.” If the Customer group is changed in the future, all user definitions using that group will use the changed values from the modified group definition. For instance, if an additional virtual directory is added to the Customer group, all users based upon that group will have access to that new virtual directory.

■ Adding and Maintaining Users

User accounts can be added or changed at any time by using the Setup Server, FTP, [Groups/Users](#) menu described on page 37. The actual process of creating or maintaining a user account is almost identical to creating or maintaining a group definition. The only difference between the two types of definitions is that a user has a password field associated with it while a group definition does not.

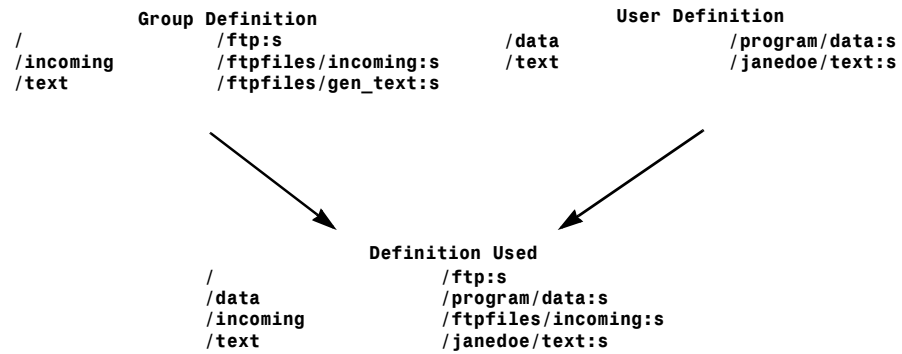
Users		
User Name:	JaniceDoe	
Group Name:	Customer	
Home Directory:		
Login Msg Text:		
Hide Hidden:		
Password:	*****	
VIRTUAL	PHYSICAL	PERMISSIONS
/	/ftp:s	RLS
/incoming	/ftp/incoming:s	WLCS
/janice	/customer/sysgy:s	RWLCDMS

When creating a new user account definition, consider whether it can use a group definition. If it can it will simplify the creation and the maintenance of that user. When a group definition is referenced in the “Group Name” field, then all of the groups fields and virtual directories are used as the initial starting point for the user definition. You may only have to enter the “Password” field information for the user.

When a user account is part of a group, the fields “Home Directory,” “Login Msg Text” and “Hide Hidden” may be left blank. If any of these fields are undefined (blank) in the user account definition when a remote user logs in with this user account, the value for the field is inherited from the current group definition.

The virtual directory portion of a user account definition that is part of a group is somewhat similar. Virtual directory definitions for a user are merged with the virtual directory definitions from the group definition. Identically named virtual directories in the user definition replace the virtual directory definition from the group definition.

For instance:



In this example, the root and /incoming virtual directories are defined by the group definition, the /data and /text virtual directories are defined by the user definition with the /text directory definition replacing the group-defined /text directory definition.

Of course, when the user is not part of a group, there are no inherited values available and no virtual directory definitions to merge with. For users that are not part of a group you must define the “Home Directory” field and you must define a virtual root directory that is the same as the home directory.

Note that field values are inherited from and virtual directories are merged with the group definition at the time that the user logs in, not the definition at the time the user account was added or last changed. This is important because it is the main reason why group definitions make account maintenance so much easier.

If you have a hundred user accounts defined and you want to give them all access to some new directory that you have created, you only have to change the group definition for those users, not each individual user account definition.

■ Using Custom Messages

The FTP server can display special messages in several situations:

- ▶ When a user connects to the server
- ▶ When a user logs onto an account
- ▶ When a user changes the current working directory
- ▶ When a user disconnects from the server

By creating and enabling these various message capabilities, you can inform and help your users by giving them information specific to your server, their account, or the files available on your system.

Not all FTP clients will display these custom messages, however most non-graphical clients will (this includes the THEOS FTP client). Graphics-based clients such as WS_FTP may display the messages in a “log window.” Web-browser FTP clients such as Microsoft’s Internet Explorer and Netscape Navigator may display some or none of the messages. In general, web-browser clients will not display the disconnect messages.

An example usage of custom messages:

```

>ftp mysite.mycompany.com george password
Connecting to mysite.mycompany.com

Sign-on message { Welcome to the MyCompany FTP site.
                  The local time is 04:34 PM on Friday, January 23, 1998.
                  You are connecting from 192.168.100.1 on George.mycompany.com.
                  Help is available by e-mailing ftpmaster@mycompany.com.
Login message  { -----
                  Hello George. Glad to see you back.
Directory change message { This is the root directory for FTP file transfers.
                           incoming      Used for uploading files to the server
                           test1        A directory used for testing transfers
                           test2        Another directory for testing transfers
                           -----
Sign-off message { FTP? bye
                  Thank you for visiting the MyCompany FTP site George.
                  You were connected for 3 minutes and 42 seconds.
                  Files uploaded: 0 (0 bytes).
                  Files downloaded: 0 (0 bytes).
>

```

■ Sign-on and Sign-off Messages

There are separate sets of messages that can be displayed when a user first connects with the FTP server and when they disconnect from the server. These messages are very useful to welcome the user to your site, let them know what the general contents of the site are, tell them how to get help, *etc.*

The sign-on and sign-off messages are created by using the **Setup Server** command, “FTP Configuration,” “[Messages](#)” menu. When defining these messages, remember that the sign-on messages are displayed before the user logs in. That is, the user name is unknown when the message is displayed. The sign-off messages are displayed just before the connection between the server and client is terminated.

The messages may use the message variables described in Appendix E: “[FTP Message Variables](#)” on page 135. For instance, the sign-on and sign-off messages displayed in the above example were defined as:

Example Sign-on Message

```
Welcome to the MyCompany FTP site.

The local time is %time on %date.

You are connecting from %ipnum on %ipname.
Help is available by e-mailing %help.
```

Example Sign-off Message

```
Thank you for visiting the MyCompany FTP site %name.
You were connected for %tconm minutes and %tcons seconds.

Files uploaded: %fup (%bup bytes).
Files downloaded: %fdwn (%bdwn bytes).
```

Note that both sets of messages are terminated with a blank line. This ensures that the messages are separated from any commands or prompts displayed following the message.

When creating the messages, remember that most users will have a console width of 80 characters or less. Try to make sure that your text lines are less than this width, even after the variables are replaced with text.

Also, remember that not all users will be able to see your messages. Therefore, do not put important information in them that they will not be able to find elsewhere, in a downloadable file.

■ Login Message

As described above, the sign-on and sign-off messages are displayed for all users. You can also define messages that are displayed for a specific user by using the login message feature. A login message is a text file that is displayed after the user has logged on and after their home directory is located.

The login messages are maintained in text files which may be located in multiple directories or in a single, fixed location. If your users have their own directories on your system, you can put the login message file in their directory.

To create a login message you must perform two steps:

1. Create the message file
2. Specify the location and name of the file for the user

• Creating Login Message Files

A login message file must be an ASCII text file. Use a text editor such as WindoWriter to create the file.

The content of the file may be anything that you want displayed when the user logs in. To make the file more readable, use one blank line at the top of the file and one blank line at the end. This separates the message text from the surrounding commands and replies displayed on the FTP client screen.

Messages may be static or you may use the message variables described in Appendix E: “[FTP Message Variables](#)” on page 135. One example that uses the variables would be a login message file for FTP administrator. This file would display some of the server statistics:

```
The current time is %time on %date.

The maximum number of users allowed is %maxusers.
The maximum number of anonymous users allowed is %maxanonymous.

There are currently %unow users connected to the server.
There have been %u24h users connected today.
A total of %uall users have connected since installation.
```

Another example is a special login message file for “anonymous” users. You may wish to tell anonymous users that their access is restricted.

For example:

Anonymous, or guest logins are allowed. However, you may only upload files to the "/incoming" directory. You may not download from this directory.

Other directories may be used for downloading, but you are not allowed to upload or change any files in those directories.

• Specifying the Location of Login Messages

The location of a user's login message file is defined in the “[Group Definition](#)” and “[User Definition](#)” screens of the Setup FTPD command.

If the file is in a single, fixed location, specify the name including the complete path specification, as illustrated below.

Users		
User Name:	JaniceDoe	
Group Name:	Customers	
Home Directory:	/ftp:s	
Login Msg Text:	/login.messages/janice.doe:s	
Hide Hidden:	YES	
Password:	*****	
VIRTUAL	PHYSICAL	PERMISSIONS
/	/ftp:s	RLS
/incoming	/ftp/incoming:s	WLCS
/janice	/customer/sysgy:s	RWLCDMS

If your login message files are located in a user's home directory, specify the name without a path specification. For instance:

Users		
User Name:	Gerald	
Group Name:	Distributors	
Home Directory:	/ftp/distribs:s	
Login Msg Text:	general.login	
Hide Hidden:	YES	
Password:	*****	
VIRTUAL	PHYSICAL	PERMISSIONS
/	/ftp/distribs:s	RWLS
/incoming	/ftp/incoming:s	WLCS
/gerald	/gerald:s	RWLCDMS

The login message file may be defined in a “[Group Definition](#),” which will be used if a user definition doesn't define a user-specific login message file.

If you have marked the files as hidden and you want to hide the files from users performing a directory listing, you must enable the “Hide Hidden”

field for the user. This field is defined in the group and user definition screens.

■ Directory Change Message

It is very useful to display a message when a user changes to a different directory on your server. The message can give them information about recent changes to files in the directory or just a descriptive list of the contents of the directory. For instance:

```
FTP? cd download/literature
```

```
Check out the brochure on the new men's boots now in stock
```

```
Index of files:
```

broch001.pdf	Brochure on men's athletic shoes
broch002.pdf	Brochure on women's athletic shoes
broch003.pdf	Brochure on men's dress shoes
broch004.pdf	Brochure on women's dress shoes
broch005.pdf	Brochure on men's boots....NEW
order.form	Order form for faxing orders

```
Directory changed to "/files/download/literature"
```

```
FTP?
```

The directory change message is text that is displayed every time that a user changes the current working directory on the server. The message is maintained in text files which may be located in each directory or in a single, fixed location.

Using a directory change message that is located in a fixed location is not too useful. The message "Directory changed to ..." in the above example is automatically displayed by the server when the directory is changed. One possible use might be a message reminding the user what they should do after they download a file.

Of much greater usage are directory change message files that are located in each directory that the user changes to. These files can be specific to the directory in which they are located.

To create a directory change message you must perform two steps:

1. Create the message file
2. Specify the location and name of the file

• Creating Directory Change Message Files

A directory change message file must be an ASCII text file. Use a text editor such as WindoWriter to create the file.

The content of the file may be anything that you want displayed when the user changes directories. To make the file more readable, use one blank line at the top of the file and one blank line at the end. This separates the message text from the surrounding commands and replies displayed on the FTP client screen.

You may use the message variables described in Appendix E: “[FTP Message Variables](#)” on page 135. The variable %dir is particularly appropriate because it displays the directory name that the user sees. That is, it is the current virtual directory name, not the physical name of the directory in your file system.

Save the file in the directory that it describes. Although there are no conventions on the name of this file, use the same name in each directory. Some suggested names for these files are: INDEX.TXT, _INDEX.TXT, README.TXT or README.

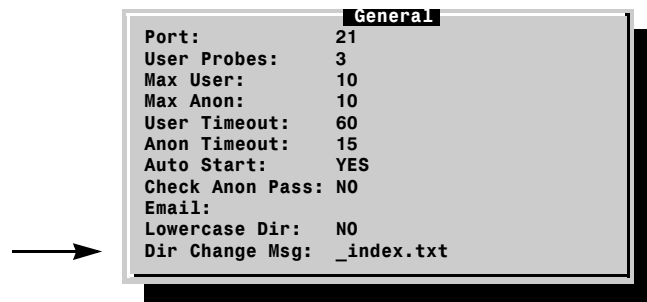
If you don’t want the users to see these files in their directory listings, change the attribute of the file to hidden.

```
>change _index.txt (h
```

The file might still be listed, depending upon the “Hide Hidden” setting in the user account setup.

• Specifying the Location of Directory Change Messages

The location of the directory change message is defined in the “[General Options](#)” screen of the Setup FTPD command.



If your directory change message files are located in each directory, specify the name without a path specification, as illustrated above. If the file is in a single, fixed location, specify the name including the complete path specification.

If you have marked the files as hidden and you want to hide the files from users performing a directory listing, you must enable the “Hide Hidden” field for the user. This field is defined in the group and user definition screens.

Groups		
Group Name: Customers		
Home Directory: /ftp:s		
Login Msg Text:		
Hide Hidden: YES		
VIRTUAL	PHYSICAL	PERMISSIONS

■ Anonymous Users

Anonymous, or guest users are treated specially by FTP servers because the user is unknown. Non-guest users are trusted to some extent because they are limited to people that you have told which user name to log onto and provided them with a password for logging onto that account.

The THEOS FTP server will allow a user to connect as an anonymous user only if the “Max Anon” value is non-zero and there is a definition for a user named “anonymous.” The “Max Anon” field is set in the Setup FTPD, “[General Options](#)” menu.

→

General	
Port:	21
User Probes:	3
Max User:	10
Max Anon:	10
User Timeout:	60
Anon Timeout:	15
Auto Start:	YES
Check Anon Pass:	NO
Email:	
Lowercase Dir:	NO
Dir Change Msg:	

You may want to enable the “Check Anon Pass” field if you want to validate passwords entered by guest logins. When this item is enabled and a user connects as an anonymous user, any password entered is checked to see if it looks like a valid e-mail address, which is the standard password used

for anonymous account logins. If the password does not look like an e-mail address they are still allowed to log in but a message is displayed informing them that the password is not valid.

When an anonymous login occurs and you have enabled logging (see “[Logging User Activities](#)” on page 61) with security logging enabled, the anonymous login is logged along with the password that they used. For security reasons, the password is not logged for normal user logins.

To create an anonymous user definition, use the Setup FTPD, “[Groups/Users](#)” menu, “[User Definition](#)” submenu:

The screenshot shows a window titled "Users" with a tab for "Anonymous". The configuration fields are as follows:

User Name: Anonymous		
Group Name:		
Home Directory: /ftp.files:s		
Login Msg Text: /ftp.files/anon.login		
Hide Hidden: YES		
Password:		
VIRTUAL	PHYSICAL	PERMISSIONS
/	/ftp.files:s	RLS
/incoming	/ftp.files/incoming:s	WLCS

Although the user name must be spelled “anonymous” the casemode of the name is not important. Do not specify a “Group Name” or a “Password.” These fields are ignored when a guest user logs in.

Depending upon the directory structure that you might have created for FTP users (see “[Controlling Access to Your File System](#)” on page 69), you may want to specify a special “Home Directory” for guest users.

When defining the virtual directories for the anonymous user account, remember that they are anonymous. That is, they may be anybody, friend or foe or just a curious user. Only define virtual directories for directories that you will not mind if a stranger downloads the files. Also, you should not give write permission to any of the directories or possibly only to one special “incoming” directory.

Review each of the permission settings for the virtual directories to make sure that you are not giving permission to do undesirable actions such as deleting files or directories.

■ Testing the Server Configuration

You can test your setup, custom messages and user definitions quite easily by using the FTP client on your system. Enter the command:

```
>ftp localhost
```

If your client is normally configured to use a proxy server, use the command:

```
>ftp localhost (noproxy
```

Localhost is a special domain name that refers to the server on the same system as the client. To test the user definitions that you have created, specify the user name and password on the command line. Because passwords are case-sensitive, you should enclose the password in a pair of quotation marks.

```
>ftp localhost user "password"
```

Assuming that your server is started and that you specified a valid user name and password, this command connects to the FTP server and logs into that user account:

```
>ftp localhost test "example" (noproxy
Connecting to THEOS_Server (192.168.100.1)
Welcome to the XYZ FTP site.
```

```
The local time is 09:58 AM on Friday, January 16, 1998.
```

```
You are user number 1 to connect today.
There is a total of 1 users connected now.
```

```
You are connecting from 192.168.100.1 on THEOS_Server.XYZ.com.
Help is available by e-mailing ftpmaster@XYZ.com.
```

```
-----
User TEST logged in.
```

```
This is the root directory for FTP file transfers.
```

```
incoming          Used for uploading files to the server
test1             A directory used for testing transfers
test2            Another directory for testing transfers
```

```
-----
FTP?
```

When testing your user definitions, look in all of the directories that are available to the logged-in user and make sure that the proper directories and files are available to the user and that inappropriate directories and files are not available or visible, as you intended.

Part III ---

THEOS HTTP Server

6 Using the Web Server

■ What is an HTTP or Web Server?

A server is a special program running on a machine that is connected to a network. The term **server** refers both to the program and to the machine that it is running on. Server software constantly checks the traffic addressed to the server machine from the network looking for requests to the server to do something.

An HTTP server looks for and processes requests for documents to be delivered to the user agent making the request. A **user agent** is a client application. For HTTP documents, the user agent is almost always a graphical or text-based web **browser** client such as Microsoft's Internet Explorer®, Netscape's Navigator® or Lynx software.

When a user on the network wants to load your page into their browser, they type in the address and name of the document. This address and document name is called a **URL** or Uniform Resource Locator. For instance,

`http://www.myserver.com/some.page`

This URL specifies that it is requesting the document named “some.page” from the server with an IP address of “www.myserver.com” and it wants the page delivered using HTTP.

Through the magic of TCP/IP and networking software, this request is delivered to your HTTP server. The server receives this request and, if it can locate the document in its virtual directories, it sends the document back to the requesting user agent. It may do other processing on the document before delivering it to the client (see “[Dynamic Web Pages](#)” on page 109) but, in most cases, it delivers an exact copy of the document exactly as it appears on the server machine.

In order for the server to do its job it must be running on a machine connected to a network.

■ Starting and Stopping the Server

There are two ways to start the THEOS HTTP server: automatically and manually.

■ Starting the HTTP Server Automatically

To start the HTTP server every time that this system is booted, enable the “[Start Server at Boot](#)” item in the Setup Server, HTTP, “[Service Options](#)” menu.



Use the **[F10]** key to save your change. The next time that this system is booted, the HTTP server is started automatically.

When the server is started automatically, there are no startup environment variables defined and CGI applications cannot utilize this feature.

■ Starting and Stopping the HTTP Server Manually

To start the HTTP server manually, use the NET START command:

```
>net start webserver
Task started as process #52.
```

The HTTP server is stopped with the NET STOP command:

```
>net stop webserver
```

As a convenience, the HTTP server can be started and stopped with various synonym names:

```
>net start www
>net start web
>net start http
```

■ Startup Environment Variables

When the HTTP server is started at any time other than during the system boot process, any environment variables defined in the starting partition are copied to the HTTP server's background partition. These startup environment variables are then available to all CGI applications invoked by the server.

For example:

```
>set "company=THEOS Software Corporation"

>set "webmaster=webmaster@theos-software.com"

>net start www
```

In this situation, BASIC language CGI applications can use the `sys.env$` function to get and use the values assigned to the environment variables "company" and "webmaster."

If the HTTP server is started automatically or if these environment variables are not defined when the server is started, the variables are not available to CGI applications and requests for them will always return an empty string.

■ Using the HTTP Server with Dial-Up Networking

If your HTTP server system accesses the Internet with a dial-up networking connection, that connection must be made by you before anyone can access your server. This is done with the `DialNet` command described in the *THEO+Net Installation and Reference* manual. For instance:

```
>dialnet start myisp
Profile "MYISP" connected successfully,
Host IP address: 206.163.58.53
Remote IP address: 204.245.231.131.
```

■ Limiting Access by IP Number or Domain Name

Whether or not your system is connected to the Internet, you may want to limit your HTTP server so that only certain sites on the network are allowed to access the server. For instance, the local network may have sites located in the administration office and other sites in the production area and you only want the sites in the office to be able to use this server for web access purposes. If your system can be accessed from the Internet, you may have even greater reasons to want to limit access to the server.

With the allow and deny lists that you can define with the “[Network Security](#)” menu, you can restrict access to specific groups or specific machines on the network, even if the user on that machine knows of a valid user account and password. These lists define which IP addresses are specifically allowed or denied access to this system’s HTTP Server.

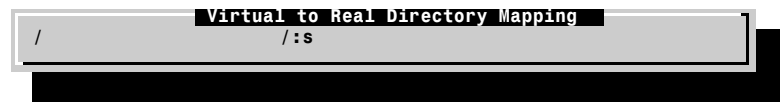
Refer to “[Allow IP and Deny IP Lists Limit Access to the Server](#)” on page 57 for a discussion of these IP address lists.

As initially installed, the HTTP server has no IP access restrictions. Thus, all sites may access the server and none are denied.

If your system has access to the Internet, the allow and deny lists should be used because an Internet connection might allow anybody to connect to your HTTP Server. Refer to Chapter 4 “[Server Security](#)” starting on page 55 for additional information about security issues and the Internet.

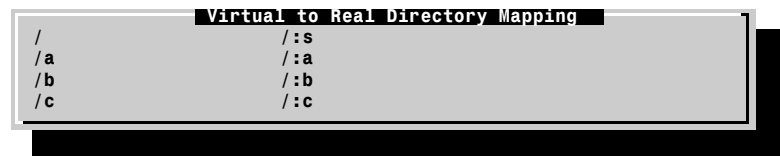
■ **Controlling Access to Your File System**

If you wanted to, you could configure your HTTP server to allow all users to access all of the files on your system. This is done by not defining any restrictions in the “[User Security](#)” configuration screen and defining a virtual directory that matches your system’s root directory. For instance:



Virtual to Real Directory Mapping	
/	/:s

Or, if the system has multiple drives and you wanted to allow access to all of them:



Virtual to Real Directory Mapping	
/	/:s
/a	/:a
/b	/:b
/c	/:c

When a user connects to your HTTP server they would have full access to the files owned by the SYSTEM account on the s drive and the other drives specified in the virtual directory definitions. In almost all cases, this is not a desirable situation.

Generally, you want to provide a user with access to only certain directories on your system. With the THEOS HTTP server, restricted access is provided in two ways: by limiting which directories they can access with

virtual directory mapping and by defining user names and passwords for specific directories and documents.

■ Virtual Directories

A virtual directory defines the relationship between what browsers can access and the names they use for directory references and what the actual directories are. For instance, the real directory structure on your system might be:

System account:	Database account:
/cgi	/data
└─source	/programs
/document	└─doc
└─pages	└─helps
/fax	└─messages
└─example	└─source
/httpicon	
/include	
/samples	
/theomail	
└─private	
└─public	
/www	

But, when a customer connects to the server, you might want them to only be able to directly access the following directory structure:

/	
└─Documents	<i>files in “/document/pages”</i>
└─└─Manuals	<i>files from “database\programs/doc”</i>
└─└─Helps	<i>files from “database\programs/helps”</i>
└─Samples	<i>files in “/samples”</i>

With virtual directory mapping, this apparent directory structure can be easily defined, even though it groups together directories from different accounts.

Virtual to Real Directory Mapping	
/	/www:s
/Documents	/document/pages:s
/Documents/Manuals	Database\programs/doc:s
/Documents/Helps	Database\programs/helps:s
/Samples	/samples:s

If a real directory is not mapped to a virtual directory and it is not a subdirectory of a directory that is mapped to a virtual directory, browsers cannot directly access the documents and files in that real directory.

“Directly access” means that the browser requests a document by specifying its URL. Any document that the browser can access can refer to and use files in directories that are not in one of the virtual directories. For instance, if a document invokes a CGI script program, that CGI script program can access any of the directories and files on your system. Or a document can reference a file with an SSI directive using the `file=` mode of the directive.

A document cannot refer to a file with a URL if the file is not in one of the defined virtual directories, and a document cannot use an SSI directive using the `virtual=` mode of the directive unless the file is in one of the virtual directories.

Files in the directory identified in the “Resources” field of the “Directory Setup” screen for the HTTP configuration (see page 49) can be accessed by the browser and any document used by the browser can refer to files in that resources directory by using the special directory name of “!icon”. This directory never appears in a directory browse view of any virtual directory.

■ Controlling Access to Specific Directories

You may want to have some directories that are accessible but only to certain people. This can be accomplished with the “User Security” menu of the HTTP configuration (see page 50). To use this feature you must:

- ▶ Use the “Virtual Directories” menu to define a virtual directory mapping to the restricted directory.
- ▶ Use the “User Security” menu to specify that that virtual directory is password-protected.
- ▶ Define the user names and passwords that are allowed access to the directory.
- ▶ Restart the server. (Changes to the HTTP configuration only take effect the next time that the server is started.)

For instance, if a directory named “/DOCUMENT/SPECIAL:A” contains special web documents that you only want your authorized customers to access, you could define a virtual directory:

Virtual to Real Directory Mapping	
/	/www:s
/Documents	/document/pages:s
/Documents/Manuals	Database\programs\doc:s
/Documents/Helps	Database\programs\helps:s
/Special	/document/special:a

Then, using the “User Security” menu:

Password Protected Directories		User names
/Special	Customer files and programs	Customer

Selected Path and Area	
Path: /Special	
Area name: Customer files and programs	

Selected User Name	
User name: Customer	
Password: Open99Sesame88Please	

If you tell your customers the user name and password that you assigned to the directory for them, they can access the documents in that directory.

For information about assigning and changing passwords, refer to the “Selecting Passwords” section in the “*THEO+Net Installation and User’s Guide*” manual, Chapter 5 “Network Security.”

When the user tries to access one of the documents in a protected directory, they are asked to identify themselves. Each brand of browser asks for this information in its own way, but essentially it will appear as:

Enter your authorization for:		OK
Resource: <i>area-description</i>		Cancel
User name:	<input type="text"/>	
Password:	<input type="password"/>	

The “*area-description*” displayed will be the information entered in the “Area name” field of the “User Security” screen. The user name is case-insensitive and must match the text that you specified in the “User name” field. The password is case-sensitive and must match exactly the text that you specified in the “Password” field.

■ Controlling Search Engine and Robot Access to Web Site

If your server is connected to the Internet and it has a registered domain name, it is possible that various search engines will visit your site to search and index the pages that it finds. You may find this desirable and even beneficial to your company.

However, if you do not want your site searched or you do not want some of the directories or pages on the site to be searched, there is a mechanism agreed to by most of the search engine providers. This mechanism is referred to as the Robots Exclusion file.

Robots (also called wanderers or spiders) are programs that traverse many pages in the World Wide Web by recursively retrieving linked pages. These programs are used by web search engines such as Lycos, Excite, Yahoo, *etc.* to find and index the words and phrases used on each of the pages that can be found on the World Wide Web. There are various reasons why it might not be desirable to have a robot access a document or set of documents or even an entire site.

In 1994 a consensus was reached between robot authors and others that essentially stated that, if a robot found a file named “robots.txt” in the root directory of a site, it would use the information in that file to decide which directories or documents would be excluded from any indexing operation. Note that this is not a formal standard but most search engines using robots conform to the agreement.

To implement this Robots Exclusion, create a file in your server’s root directory named ROBOTS.TXT. For instance, if your virtual root directory (see “[Virtual Directories](#)” on page 49) is mapped to the real directory /www:s, then you would create the file /www/robots.txt:s. The robots.txt file is an ASCII stream file that contains two or more lines of text specifying which robots are excluded and which directories and files they are excluded from.

The file consists of two types of records, each formatted similarly:

field: value comment

Comments can be included in a line with the “#” character. Any characters following and including the “#” character are ignored.

The file starts with one or more lines with a *field* of [User-agent](#), followed by one or more lines with a *field* of [Disallow](#).

```
# Exclude all robots from all pages
User-agent: *
Disallow: /
```

or

```
# Exclude all robots from some directories
User-agent: *
Disallow: /personal/pages/
Disallow: /temp/ # temporary files
```

User-agent . The *value* of this field is the name of the robot that this record is describing the access policy for. If more than one User-agent field is present the record describes an identical access policy for more than one robot. At least one field needs to be present per record.

If the *value* is “*” the record describes the default access policy for any robot that has not matched any of the other User-agent records. This is the standard User-agent record used in most robot.txt files.

```
User-agent: *
```

Disallow . The *value* of this field specifies a partial URL that is not to be visited. This can be a full path or a partial path. Any URL that starts with this *value* will not be retrieved. For instance,

```
Disallow: /help
```

disallows both /help.html and /help/index.html, whereas

```
Disallow: /help/
```

would disallow /help/index.html but allow /help.html.

At least one Disallow field needs to be present in a file. You may have several:

```
User-agent: Scooter                # AltaVista
User-agent: ArchitextSpider        # Excite
User-agent: ExciteSpider           # Excite
User-agent: Lycos_Spider           # Lycos

Disallow: /personal/pages/
Disallow: /temp/                   # temporary files
Disallow: /customer/default.asp    # customer main page
```

Note that, when a specific document is disallowed, other documents in the directory path might be indexed. For instance, in the above example, the document default.asp in the directory /customer is disallowed. If there are any links to other pages in this directory, a robot might find and index those pages. The link might be in a page on your site or a link from a page on someone else’s server that the robot indexed.

The casemode of the URL is significant

As mentioned at the beginning of this subject, the robots.txt file is recognized and used by most robot indexers, but not necessarily all.

■ **Client Access**

User agents, or browsers, send requests to the HTTP server. These requests are of two types:

- ▶ GET a document
- ▶ POST information from a form

When the user agent requests a document the request may specify a specific document or it may be a request with a URL that identifies a directory path but does not specify a specific document in that directory. If the request identifies a specific document, the server delivers that document. For instance:

```
http://my.server.com/document/manuals/current.html
```

This request sends the `CURRENT.HTML` document to the client.

■ **Default Documents**

The request to the server does not have to include the document name:

```
http://my.server.com/document/manuals/
```







When the server receives this request it uses the “[Default Documents](#)” field of the “[Service Options](#)” screen for the HTTP configuration (see page 46) to locate the default document in the directory requested. For instance, if the default document specification is “`default.*,index.*`” the server will search in its virtual directories for “`/document/manuals/default.*`” and, if at least one file is found it uses the first file found. When no files are found matching that specification it searches for the next default document specification (“`index.*`” in this example), and so on.

The default document specification may use wild cards in the file-name or file-type as shown in the above example or the specification can use specific file names. When the file-type is specified with a wild card backup files are ignored. That is, files with a file-type of “`.backup`” are not included in the search. However, if files of this type are specified explicitly in the “[Default Documents](#)” field, they are delivered.

■ Directory Browsing

If no files matching the “[Default Documents](#)” specification can be found in the directory requested, the server might display the files in the directory in a directory browse view.

When the “[Directory Browse](#)” field (see page 47) is set to “Yes” and no default document is found in the directory requested, the HTTP server performs a FileList of the directory, formats the directory as a web document and presents that document to the browser.

Index of /test			
<i>switch to plain format</i>			
Name	Last Modified	Size	Description
 Parent Directory	09 Jan 1998 02:41p		
 counter.stm	12 Jan 1998 03:56p	0.2K	
 form.stm	23 Jan 1998 02:51p	0.3K	
 hr.stm	21 Jan 1998 01:52p	1.1K	
 page.stm	23 Jan 1998 08:51a	1.4K	
 phrase.stm	23 Jan 1998 12:21p	1.0K	

The plain format display is similar except that it uses a monospace font.

The directory browse view will not include some files in the directory. Specifically, libraries, files with the hidden attribute set, backup files and files with a file name of “_HEADING.” “_FOOTING.” or “_DESCRIP.” are skipped.

The icons on the left and the file names are hyperlinks to the file. The icon displayed for each file is chosen from the icon files in the directory specified in the “[Resources](#)” field of the “[Directory Setup](#)” screen for the HTTP configuration (see page 49). The specific icon used for a file is determined by the file association of its file-type.

These icons can be customized. Refer to “[Customizing Common Resources](#)” on page 105.

If “[Directory Browse](#)” is set to “No,” the server refuses access to the directory listing and presents the 413 error message to the browser instead.

■ Customizing the Directory Browse View

The text used in the default directory browse view is defined in the `SYSTEM.THEOS32.HTTPDCFG` file in the section labeled `[Literals]`.

```
[Literals]
Index=Index of %s
Plain=Switch to plain format
Table=Switch to table format
Name=Name
Modified=Last Modified
Size=Size
Description=Description
Parent=Parent Directory
```

These items may be edited if you wish to have different labels for the column headers or you need to translate the display to a language other than English. Open the file with an editor such as WindoWriter and change the text following the equal sign for each item. Do not change the text before the equal sign because the server needs that text to identify each literal.

The directory browse view is suitable for most directory displays. However, you may want to customize the information displayed above the directory listing and below it, and you may wish to define content descriptions for some or all of the files in a directory. You may use these custom features by creating special files in the directory.

If either or both of the following files exist in a directory, they are used instead of the default text:

_HEADING. If this file exists in a directory, its contents are used as the beginning of the directory browse document for this directory.

This file's content is responsible for the beginning of the document and must include the HTML elements for the start of a document.

For instance:

```
<HTML>
<HEAD>
<TITLE>Directory of /www/browse</TITLE>
</HEAD>
<BODY>
<H1 align=CENTER>Directory Heading</H1>
<CENTER>
<A href="./?table">Table View</A> /
<A href="./?plain">Plain View</A>
</CENTER>
<P>
Additional text displayed before directory listing...
```

The two anchors in the example (<A href=...) are hypertext links to the alternate displays for this directory.

The above example is only an example. You may use any of the HTML elements and text desired as long as it starts with <HTML> and does not contain a </BODY> or </HTML> element which would terminate the document contents.

The file can be used to describe the directory's purpose and usage, give instructions for using the directory listing, or whatever information you want displayed when this directory is displayed in the browse view.

The file may use the Server Side Include (SSI) directives described in Appendix F: "[SSI Directives](#)" starting on page 137.

_FOOTING. When this file exists in a directory, its contents are used as the end of the document for this directory browse listing.

This file's content is responsible for the ending of the document and should include the HTML elements for the end of a document. For instance:

```
<P>Additional HTML BODY elements for directory listing...
</BODY>
</HTML>
```

The file can be used to provide additional information to the user about the directory.

The file may use the Server Side Include (SSI) directives described in Appendix F: "[SSI Directives](#)" starting on page 137.

Another file can be used to customize the directory listing itself by providing descriptions of individual files in a directory.







_DESCRIP. The contents of this file will be used to provide the text for the “Description” column of the directory listing. It should contain one line for each file that you want described in this directory. Each line contains two elements separated by a comma: the file name and the description of that file.

```
customer.master, Master database for customers
invoice.detail, Line item data for each invoice
private., Subdirectory containing private information
```

The contents do not have to be sorted.

The descriptive text should be kept short but, if you do need a long description for a file or directory, use it and it will display properly in the “Description” column, using more than one line if necessary.

If all three of the file examples were defined in a directory without one of the default documents, it would be displayed as:

Directory Heading			
<i>Table / Plain</i>			
Additional text displayed before directory listing ...			
Name	Last Modified	Size	Description
 Parent Directory	09 Jan 1998 02:41p		
 customer.master	12 Jan 1998 03:56p	0.2K	Master database for customers
 customer.detail	23 Jan 1998 02:51p	0.3K	
 invoice.master	21 Jan 1998 01:52p	1.1K	
 invoice.detail	23 Jan 1998 08:51a	1.4K	Line item data for each invoice
 orders.master	23 Jan 1998 12:21p	1.0K	
Additional HTML BODY elements for directory listing...			

■ Customizing Common Resources

As initially installed, the default directory identified in the “[Resources](#)” field of the “[Directory Setup](#)” screen for the HTTP configuration (see page 49) is “/httpicon:s” and contains a set of icon files and document pages used for standard error messages. These files can be customized by replacing them with your own files.

This directory is also the recommended location for adding your own set of icons that are used in your web documents. Add one or more subdirectories to this /HTTPICON directory to avoid conflicts with these server icons. For instance, with a new subdirectory of “/HTTPICON/MYIMAGES” your documents can use references of:

```
<IMG src="/!icons/myimages/some.gif">
```

■ File Icons

The icon files are files with a file-type of “gif” and are used by the directory browse display described in “[Directory Browsing](#)” on page 101. You can replace these files with your own icons for the various file-types desired.

The file-name for the icon must match the associated file-type of a file for it to be used for the file. Specifically:

File Association	Icon File
text/html	HTML.GIF
application/msword	WORD.GIF
application/mswrite	TEXT.GIF
application/pdf	PDF.GIF
application/x-compress application/x-tar application/zip	ZIP.GIF
application/rtf text/richtext text/rtf	RTF.GIF
audio/*	SOUND.GIF
video/*	VIDEO.GIF
text/*	TEXT.GIF
application/octet-stream	BINARY.GIF
other	UNKNOWN.GIF

There are a few icons defined and used that do not match a file's associated file-type:

- blank.gif** A transparent, single-pixel image used as a placeholder in the directory browse heading line.
- folder.gif** The icon used for directories and subdirectories.
- parent.gif** The icon used for the hyperlink to the directory that is a parent of the current directory being displayed.
- unknown.gif** An icon used when no other icon matches the file association of the file.

When replacing icons in this set use an image size of 24×24 pixels.

■ Error Message Documents

When the server detects an error when trying to process or deliver a document, it looks in the resources directory for a file named “*errnum.asp*” where *errnum* is the number of the server error. For instance, when a request is made for a document in a protected directory (see “[Controlling Access to Specific Directories](#)” on page 96) and the user doesn't supply the proper authorization for that directory, the server reports the error with a 401 error message. If it can find the file named “401.ASP” in the resources directory it processes that file and delivers it to the client.

If it cannot find an error message file for the error number it reports the error to the client using a standard MIME header identifying the error number and a standard English message for that condition.

You can add additional error message files to this directory or modify the existing files. For instance, you might want to translate them to a language other than English or you might want to provide specific contact information for reporting the error to you via e-mail or you might want to add your company name and logo to the document, *etc.*

Note that these error documents have a file-type of “asp” meaning that they are Active Server Pages and you can use any of the ASP statements described in Appendix H: “[ASP Reference](#)” starting on page 189. ASP documents may also use most of the SSI directives described in Appendix F: “[SSI Directives](#)” starting on page 137.

When these error documents are invoked the server will have already defined some variables that the ASP statements in the document may use. Specifically, the following variables are defined:

Variable	Value
ErrorNumber	Error number. For instance, 301, 302, <i>etc.</i>
Error	
URL	URL of request generating the error
Home	URL of server's home page
Location	Only used for error number 302. Contains the URL of the moved-to location.

The standard errors that might be reported by the server include:

Error Code	Meaning
301	Document moved permanently
302	Document moved temporarily
304	Not modified
400	Invalid request
401	Unauthorized access
404	Document not found
500	Internal server error
501	Not implemented
503	Service unavailable
505	HTTP version not supported

7 Dynamic Web Pages

Most web sites found on the Internet are collections of static pages. That is, the documents are text files delivered to the user's browser when requested and do not change unless someone replaces the text file with a different version of the information. Dynamic documents can also be delivered to the user's browser.

A dynamic document is a page that is or can be different each time that it is retrieved. These differences may be as simple as a counter that changes each time that the page is retrieved through a completely different document that is delivered.

There are four basic methods of creating dynamic documents:

- ▶ Document uses Server Side Include (SSI) directives
- ▶ Document uses Active Server Page (ASP) statements
- ▶ Document uses Common Gateway Interface (CGI) application
- ▶ Document uses client-side JavaScript functions

Documents using SSI and ASP features are static document files that are interpreted by the HTTP server and processed prior to delivering the resulting document to the user's browser. A CGI-generated document is a document that is created at the time the CGI application is invoked.

A document may use various elements from multiple methods to create a dynamic document. For instance, a document may use SSI directives to include a counter and the server's current date in a document and use a CGI application to generate the contents of a table or it may use ASP statements to test the browser's capabilities to determine which type of HTML features to use in the document.

There are one limitation to this intermingling of dynamic document methods:

- ▶ CGI applications cannot generate documents using ASP statements. However, a document may use the SSI `#exec` directive to invoke a CGI application to include the output from the CGI script into the SSI document.

■ **Server Side Includes (SSI)**

A Server Side Include is a directive that is coded into a normal web document. This directive, when “server-parsed” by the server before delivering the document to the user agent (browser), is interpreted and is replaced by the value or text specified by the directive. A common example of a SSI document is one that includes a counter telling you that “You are visitor number ...” or “Welcome, the current time is ...,” *etc.* These counters or times are determined in realtime at the time that the server delivered the document to the user agent.

As mentioned briefly at the beginning of Chapter 6 “[Using the Web Server](#)” starting on page 91, the server normally delivers a copy of a document without modifying it in any way. Because it takes time and resources to parse a document, and because many documents contain no dynamic content, documents are not automatically examined for SSI directives or ASP statements. The server determines whether or not it should parse a document based upon its file-type.

A document is parsed before delivery if:

1. The file-type of the document is “SSI” or “ASP.”
2. The file-type of the document is one of the file-types specified in the server’s configuration in the field [SSI Files](#) of the [Service Options](#) menu.

When either of these conditions are true, each line of text in the document is examined for special SSI directives. If either are found they are processed and replaced with the results of the directive or statement.

■ **Server-Side-Include Format**

SSI directives are specified within an HTML comment line. A comment line in an HTML document looks like this:

```
<!--This is a comment-->
```

The “<!--” and “-->” define the beginning and ending of the comment. Comments are not displayed by the browser that is viewing the page.

By using a special format inside of the comment, the server recognizes it as an SSI directive rather than something to skip. A Server Side Include directive looks something like this:

```
<!-- #include file="standard.text" -->
```

When processed, the entire HTML comment containing the SSI directive is replaced with the processed value of the directive. For instance, if a portion of a page contains:

```
<P>The date is <!-- #echo var="DATE_LOCAL" --> and  
you are visitor number <!-- #counter name="visitors" -->.
```

is delivered to the client as if it had been a static document containing:

```
<P>The date is Thursday April 16 1998 and  
you are visitor number 45.
```

Of course, it is not a static document and every time that it is retrieved from the server the date and visitor number is updated to reflect the current date on the server and the count of the number of times that the page has been retrieved.

The SSI directives are described in Appendix F: “[SSI Directives](#)” starting on page [137](#).

■ **Active Server Pages (ASP)**

An Active Server Page is an HTML document that contains ASP statements and expressions that are interpreted by the server prior to delivering the document to a user-agent (browser). This is very similar to a document containing SSI statements. However, SSI statements are directives with no decision-making capabilities and very limited functionality. ASP statements are programming-style statements and capabilities very similar to many of the C-language or BASIC-language statements.

Although an ASP document can perform many functions that a CGI application can, because the ASP document is interpreted it is best to limit it to simple tasks such as transferring cookie information from a browser to form fields. ASP documents can read and write records to a stream file but again, a CGI application is more efficient and flexible when updating a database.

The principle advantage of ASP documents over CGI applications is that you don't have to be a programmer to use ASP statements, merely embed them in the HTML document and let the server process them as it delivers the document to the user agent.

An ASP document may have SSI statements in it which are processed prior to interpreting any of the ASP statements in the document.

An ASP document must have a filetype of "ASP."

The ASP statements and functions are described in Appendix H: "[ASP Reference](#)" starting on page [189](#).

■ Common Gateway Interface (CGI)

The Common Gateway Interface (CGI), is a standard for interfacing programs with HTTP servers. A plain HTML document that the Web server retrieves is static, meaning that it exists in a constant state. Each time that it retrieves the document it has the same contents.

A CGI-generated document is a document that does not exist as a disk file on the server system. Instead, it is the “printed” output of a program. The HTTP server invokes the program as a task and all of the stdout output from the program is captured and delivered to the client as the web document. The contents of the document are completely under the control of the program and may be the same each time the program is invoked or very dynamic with information included from one or more databases, etc.

CGI programs are invoked in one of three situations:

- ▶ By a SSI `#exec` CGI= directive
- ▶ As the `action=` for a document form
- ▶ As a URL when it is followed by a query

In the first situation, the output of the CGI application is included in the document using the SSI directive.

CGI programs are usually used to process the output of an HTML page’s FORM. For instance, a web page may have an input form for submitting an e-mail message to the webmaster.

```
<FORM action="/cgi/mailto" method="POST">
<INPUT type="HIDDEN" name="to" value="webmaster@your-site.com">
<p>From:&nbsp;<INPUT type="TEXT" name="from" size=40>
<p>Subject:&nbsp;<INPUT type="TEXT" name="subject" size=40>
<p>Message:&nbsp;<TEXTAREA name="message" rows=10 cols=60>
</TEXTAREA>
<p align="CENTER"><INPUT type="SUBMIT" value="Submit">
&nbsp;&nbsp;<INPUT type="RESET" value="Clear form">
</FORM>
```

The `action="/cgi/mailto"` attribute of the FORM element specifies that, when the user submits this form, the program named MAILTO in the directory /CGI/ is invoked to process the fields of the form. That program will receive a set of name/value pairs:

```
to=webmaster@your-site.com
from=user input to first field
subject=user input to second field
message=user input to text area field
```

The MAILTO program must use the [CALL CGI.INIT](#) function to convert these name/value pairs into a form that is usable by the program. The [FN.CGI.GET\\$ \(name\\$ \)](#) function is used to retrieve the values for each of the names. The MAILTO program can then use the information to create a file and send the e-mail message with the [SendMail](#) command or whatever you want the program to do with the information.

CGI applications used to process form data must output something on stdout for the server to complete the process. There are basically three options for the application:

- ▶ Output a new document, possibly telling the user that the information was processed successfully.
- ▶ Output a special error or status message to the user's browser (see [CALL CGI.ERROR \(text\\$ \)](#) and [CALL CGI.STATUS \(status%, desc\\$ \)](#) functions).
- ▶ Output a special redirection message that the server uses to locate and deliver the next document file to the user (see [CALL CGI.LOCATION \(url\\$ \)](#) function).

■ CGI Program Location

All CGI programs **must** be located in the CGI directory tree specified in the server's configuration in the [Directory Setup](#) menu, [CGI](#) field. The URL reference for the program (in the SSI [#exec](#) directive or in the [FORM](#) [action=](#) attribute) must specify the complete path to the program.

Programs should have a file-type of ".COMMAND." Do not use a command library.

The CGI functions are described in Appendix G: "[MultiUser Basic Language CGI API Functions](#)" starting on page [157](#).

■ Server, HTTP and Environment Variables

Although a CGI application cannot use the SSI [#echo](#) directive to access server and HTTP variables, nor can it use ASP statements to access those variables, it does have full access to all of the server's variables. Before the server invokes a CGI application, it first creates environment variables for all of the server and HTTP variables defined at the time the CGI application is invoked.

In addition, if there were any environment variables defined in the partition that started the HTTP server (see [“Startup Environment Variables”](#) on page 93), those variables are also copied to the partition used by the CGI application.

Environment variables are accessed by a CGI application with the `sys.env$` function. A list of the server variables and HTTP variables that may be used can be found in Table 2: [“Server Variables”](#) on page 147 and Table 3: [“HTTP Header Variables”](#) on page 149.

■ Limitations

A document created by a CGI application cannot use ASP statements. It can generate files containing SSI directives. After the document is created by the CGI application it is processed by the server and any SSI directives are expanded before it is delivered to the client.

Because the document created by a CGI application does not exist as a disk file there is no “current working directory” associated with it. All hyperlinks and other references to files should be coded with a complete path specification.

The CGI application executes as a background task and does not have a console available to it. Any unexpected run-time errors will not display their error message on the “screen.”

■ Client-Side JavaScript

Client-side JavaScript is not Java. Client-side JavaScript is an interpretive language that is added to an HTML page that is delivered by the server to the client and requires no special processing by the server. Client-side JavaScript is embedded directly into HTML pages and is interpreted by the browser completely at runtime.

An HTTP server delivers content to a client without caring too much about what that content is. Because client-side JavaScript pages are normal HTML pages that contain JavaScript function definitions and references, any server can deliver the page to the client. Only the client has to have special knowledge and builtin capabilities to handle the JavaScript code.

Not all browsers support JavaScript. It was developed by Netscape Communications and their browser, Netscape Navigator version 3 and above, Netscape Communicator and Microsoft Internet Explorer version 3 and above all support JavaScript. Other browsers may support it also.

Client-side JavaScript statements embedded in an HTML page can respond to user events such as mouse-clicks, form input, and page navigation. For instance, you can use a JavaScript function to verify that a user entered valid information into a telephone number or zip code field. Without any network transmission, the HTML page with embedded JavaScript can check the entered data and alert the user with a dialog box if the input is invalid.

Form validation is one of the most common uses for client-side JavaScript. JavaScript provides a quick, easy way to check a user's input for mistakes, typing errors and the omission of required fields. Because validation takes place on the client machine, there is no delay for contacting a remote server. The user gets quicker responses and network bandwidth and server processing power are both conserved.

There are many books available that describe the JavaScript language and show you how to customize your web pages by using JavaScript. For an on-line source of information, refer to the Netscape web pages. Specifically:

```
http://developer.netscape.com/library/documentation/communicator/  
jsguide4/index.htm
```

Or, for a downloadable form:

```
http://developer.netscape.com/library/documentation/index.html
```

■ Validating Form Input

One of the more important uses of JavaScript is to validate form input to server-based programs such as CGI programs. This is useful because:

- ▶ It reduces the load on the server. “Bad data” is already filtered out before data is passed to the server-based program.
- ▶ It reduces delays caused by input errors. Otherwise, when validation is performed on the server, data must travel from client to server, be processed, and then returned to the client for reentry.
- ▶ It simplifies the server-based program.

Generally, you want to validate input in at least two places:

- ▶ As the user enters it, with an `onChange` event handler on each form field that you want validated.
- ▶ When the user submits the form, with an `onClick` event handler on the button that submits the form data.

■ Embedding JavaScript in HTML

JavaScript is used in an HTML document in the following ways:

- ▶ As statements and functions within a `<SCRIPT>` tag.
- ▶ By specifying a file as the JavaScript source (rather than embedding the JavaScript in the HTML).
- ▶ By specifying a JavaScript expression as the value of an HTML attribute.
- ▶ As event handlers within certain other HTML tags.

Unlike HTML code, JavaScript is case sensitive.

• Using the `SCRIPT` Tag

The `<SCRIPT>` tag is an extension to HTML that can enclose any number of JavaScript statements as shown here:

```
<SCRIPT>
  JavaScript statements...
</SCRIPT>
```

A document can have multiple `<SCRIPT>` tags, and each can enclose any number of JavaScript statements.

- **Specifying a File as JavaScript Source**

The SRC attribute of the <SCRIPT> tag lets you specify a file as the JavaScript source rather than embedding the JavaScript in the HTML.

For example:

```
<HEAD>
<TITLE>Some Page</TITLE>
<SCRIPT SRC="common.js">
...
</SCRIPT>
</HEAD>
<BODY>
...
```

The closing </SCRIPT> tag is required.

This attribute is useful when sharing functions among many pages.

External JavaScript files should have the file-type “js”, and the server must map the “js” suffix to the MIME type “application/x-javascript,” which the server sends back in the HTTP header.

- **Hiding Scripts within Comment Tags**

Older versions of browsers and current versions of some browsers do not recognize JavaScript. To ensure that these browsers ignore JavaScript code, you should place the entire script within HTML comment tags, and precede the ending comment tag with a double-slash (//) that indicates a JavaScript single-line comment:

```
<SCRIPT>
<!-- Hide script contents from old browsers.
JavaScript statements...
// End the hiding here. -->
</SCRIPT>
```

Since browsers typically ignore unknown tags, non-JavaScript-capable browsers will ignore the beginning and ending SCRIPT tags. All the script statements between these tags are enclosed in an HTML comment, so they are ignored too. JavaScript-capable browsers properly interpret the SCRIPT tags and ignores the line in the script beginning with the double-slash (//).

Although you are not required to use this script-hiding technique, it is considered good etiquette so your pages won’t display the raw script statements for those users with non-JavaScript-capable browsers.

8 Getting Information

The primary purpose of an HTTP server is to deliver content or information to a client. It can also receive information from the server, the client and the user of that client application. This information can be either incorporated into the content delivered to the client or used to make decisions in a dynamic document created with a CGI script or Active Server Page.

■ Information from the Server

There is a set of fields defined by the server that can be accessed and used in a document. These fields can be access with SSI directives, a CGI application or Active Server Page statements.

■ Server Variables in SSI

The server variable field values can be inserted into a document with the SSI directive `#echo` var. For instance:

```
<!-- #echo var="Server_Software" -->
```

This directive is replaced with the server software name and version as “THEO+Server/1.0.”

Refer to Table 2: “[Server Variables](#)” on page 147 and Table 3: “[HTTP Header Variables](#)” on page 149 for a description of all of the variables that can be used with this directive.

■ Server Variables in CGI

If a CGI program needs to access one of the server fields it can use the `SYS.ENV$` function to get the value of the environment variable corresponding to that server variable. Before a CGI application is invoked, all of the server variables and HTTP variables are assigned to like-named environment variables:

```
SERVER.SOFTWARE$ = SYS.ENV$(17, "SERVER_SOFTWARE")
```

The lists of variable names shown in Table 2: “[Server Variables](#)” on page 147 and Table 3: “[HTTP Header Variables](#)” on page 149 for a description of all of the variables that are assigned to environment variable names before a CGI application is invoked.

■ Server Variables in ASP

The server variable field values can be inserted into a document or assigned to an ASP variable with the ASP [Request.ServerVariables](#) object. For instance:

```
<% asp_var = Request.ServerVariable("ServerName") %>
```

This statement assigns the server name to the variable `asp_var`.

The user agent's or browser's capabilities can be accessed with the `MSWC.BrowserType` object. For instance:

```
<% Set bc = Server.CreateObject("MSWC.BrowserType")
If bc.TextOnly Then
    ...
End If %>
```

Refer to Table 10: “[Server Variables](#)” on page 222 and Table 5: “[Browser Capabilities](#)” on page 166 for a description of all of the server variables and browser capabilities that can be used with an ASP document. Remember, an ASP document is first processed for SSI directives, so server variables can also be accessed with the SSI methods described previously.

■ Types of Information Received from Client

There are several types of information that can be sent by the client. These include:

- ▶ Browser Identification
- ▶ Cookies
- ▶ Form Data
- ▶ Query String

■ **Browser Identification**

When a browser requests a document from a server it always sends a string identifying itself. This identification is available to documents using SSI directives or ASP statements.

The SSI directive that accesses this browser identification string is:

```
<!-- #echo var="USER_AGENT" -->
```

or

```
<!-- #echo var="HTTP_USER_AGENT" -->
```

The ASP statement that accesses the browser identification string is:

```
<% Request.ServerVariables("HTTP_USER_AGENT") %>
```

or as the object of an If statement or an expression.

The actual browser identification string can be very cryptic. For instance, the string sent by a browser might be:

```
Mozilla/4.03 [en] (Win95; I ;Nav)
```

This particular string identifies the browser as Netscape Navigator, version 4.03 on a Windows 95 platform.

To properly decode these browser identification strings, the server uses a lookup file named `SYSTEM.TEOS32.BROWSCAP`. CGI applications can access this decoded browser capabilities with the [FN.CGI.BROWSER\\$ \(capability\\$ \)](#) function.

An ASP document can access the decoded browser capabilities with the `Server.CreateObject` object:

```
<% Set bc = Server.CreateObject("MSWC.BrowserType")
```

■ Query String

The query string is the text following a URL specification. For instance, if the URL specification is

```
http://your.domain.com/some.page?search=some+word&a=1&b=2
```

The query string in the above URL is “search=some+word&a=1&b=2.”

Although query strings can be entered by the user when requesting a specific document, they are normally supplied by a hypertext link in a document or more commonly by forms submission. Query strings can be accessed by SSI directives and used in CGI scripts and ASP documents.

■ Query Strings and SSI

The value of the query string sent when a document is retrieved can be accessed with the following directives:

```
<!-- #echo var="Query_String" -->
<!-- #echo var="Query_String_Unescaped" -->
```

The first directive is replaced with the value of the raw query string while the second directive is replaced with the decoded or unescaped query string. For instance, if the `Query_String` is “search=one+two+three” then `Query_String_Unescaped` will be “search=one two three” with the plus sign replaced by the space character.

■ Query Strings and CGI

The query string can be used in THEOS CGI programs. If the query string exists because of a FORM submission using the GET method, it is best to process it using the techniques described in the section on “[Form Data](#)” on page 125. The name/value pairs defined in a query string are more easily extracted with the methods described there.

If a CGI program does need to access the entire query string it can use the `SYS.ENV$` function to get the value of the `QUERY_STRING` environment variable:

```
QUERY$ = SYS.ENV$(17, "QUERY_STRING")
```

The `QUERY_STRING` environment variable contains the value of the query string specified when the CGI application was invoked.

■ Query Strings and ASP

The value of the query string sent when a document is retrieved can be accessed with the following statements:

```
<% var = Request.ServerVariables("Query_String") %>
<% var = Request.ServerVariables("Query_String_Unescaped" ( %>
<% var = Request.QueryString("search") %>
<% var = Request.("search") %>
```

The first statement assigns the value of the raw query string to the variable while the second statement assigns the value of the decoded or unescaped query string to the variable. For instance, if the `Query_String` is “search=one+two+three” then then `Query_String_Unescaped` will be “search=one two three” with the plus sign replaced by the space character.

The third and fourth statements extract a specific, unescaped variable value defined in the query string. Both of these example statements will get the value of the variable “search” defined in the query string.

The difference between the last two statements is that the fourth example may find the variable name defined in another place first. Specifically, if the variable is defined by a cookie or form, its value will be extracted from that location without looking in the query string.

■ Cookies

A cookie is a named piece of information sent by the server to a client browser. If the browser supports cookies and the user has enabled them, the cookie is stored on the client’s system. After a cookie is stored on the client’s system, every request for a document by the client to this server causes the cookie name and value pair to be delivered to the server.

Cookies are normally used by a web page or set of related web pages for storing information provided by the user. Subsequent access to the web page can then retrieve the information without requiring the user to enter it again. For instance, a form might request the user’s name, mailing address and e-mail address. Later accesses to web pages that need to know that information can check to see if the user has supplied it already and use that saved data.

Cookies are defined with special MIME headers sent before a web page is transmitted. They can only be set by CGI applications because no other type of document can control the MIME headers sent before the HTML page.

Cookie values can be accessed by any web page using SSI directives or ASP statements and they can be accessed by CGI scripts.

■ Cookie Values and SSI

The value of all cookies maintained by the client browser can be accessed with the following directive:

```
<!-- #echo var="HTTP_COOKIE" -->
```

This directive is replaced with the cookie sent by the browser.

■ Cookie Values and CGI

A specific cookie name/value pair can be accessed and used in a CGI program by using the `FN.CGI.GET.COOKIE$ (name$)` function. For instance:

```
CUSTID$ = FN.CGI.GET.COOKIE$ ("custid")
```

The function scans the cookie sent by the browser looking for the substring “custid=”. When found, it returns the substring following this cookie name assignment string. For instance, if the cookie sent by the browser was “custid=THEOS; email=sales@theos-software.com,” the value returned by the above function call would be “THEOS.”

Cookie values are set by a CGI application with the `CALL CGI.PUT.COOKIE (name$, value$, expire, path$, domain$)` function. To use this function, you must call it after the `CALL CGI.INIT` function but before calling the `CALL CGI.BEGIN (title$, txtcolor$, bgcolor$)` function. For instance:

```
CALL CGI.INIT
...
CALL CGI.PUT.COOKIE("name", "John Doe", 99999, "/", "www.myserver.com")
CALL CGI.PUT.COOKIE("email", "jdoe@mymail.com", 99999, "www.myserver.com")
...
CALL CGI.BEGIN("Page Title", "black", "white")
...
```

■ Cookie Values and ASP

Cookies are accessed and set in an ASP document with the `Request.Cookies` object and the `Response.Cookies` object. To get the value of a specific cookie name/value pair use a statement similar to the following:

```
Custid = Request.Cookies("custid")
```

This statement scans the cookie sent by the browser looking for the substring “custid=”. When found, it returns the substring following this cookie name assignment string. For instance, if the cookie sent by the browser was “custid=THEOS; email=sales@theos-software.com,” the value returned by the above statement would be “THEOS.”

Cookie values are set in an ASP document with the `Response.Cookies` object. To assign a cookie value, you must use the object prior to outputting any part of the HTML document.

For instance:

```
<%
... ' No statements generating HTML text or HTML text itself
Response.Cookies("User") = "John Doe"
Response.Cookies("User").Path = "/"
Response.Cookies("User").Domain = "www.myserver.com"
Response.Cookies("email") = "jdoe@mymail.com"
Response.Cookies("email").Path = "/"
Response.Cookies("email").Domain = "www.myserver.com"
...
%>
<title>Example document defining cookies</title>
...
```

■ Form Data

FORMs are the standard method of sending information from the user to the server. The information collected in a form may be processed with either a CGI program or an ASP document.

■ Form Data and CGI

CGI programs are frequently used to process FORM data whether the FORM was submitted using the GET or the POST method. When a FORM is submitted with the GET method, the field names and values are sent via a query string to the program specified in the FORM action= specification. When a FORM is submitted with POST method the field names and values are sent via stdout and stdin to the program specified in the FORM action= specification.

Because FORM data is always name/value pair specifications, there are special CGI functions provided in the CGI ToolKit to access and extract these name/value pairs.

The `CGI.INIT` function is used at the beginning of the application to convert the GET method query string or the POST method stdin into a common format that is easily used by the program. To get individual values for a specific name, the `FN.CGI.GET$` function is used. For instance, with a field of “search=one+two+three” the following program code can get the value of the search name:

```
CALL CGI.INIT
...
SEARCH$ = FN.URL.DECODE$(FN.CGI.GET$("search"))
```

The `FN.URL.DECODE$` function is used to decode any special characters that might be in the field value.

■ Form Data and ASP

ASP documents are also used to process FORM data submitted with the GET method, particularly when the form is simple. A typical usage would be to accept the form data of a mailto type document. When a FORM is submitted with the GET method, the field names and values are sent via a query string to the program or document (ASP) specified in the FORM action= specification.

FORM data is always name/value pair specifications. To access the value for a specific FORM data name you must use the `Request.Form` object:

```
email_from = Request.Form("From")
email_to = Request.Form("To")
```

A: Contacting THEOS

Support for THEO+Server and other THEOS products is provided to authorized dealers and resellers of THEOS products. End users should contact their THEOS dealer regarding questions or problems relating to installation or operation of the THEOS operating system and other THEOS products.

THEOS Support Services for Resellers and Distributors are designed to provide the type of assistance best suited to your needs. Support options range from no-cost / low-cost information services on the WWW to THEOS on-site training classes, fee-based direct support or an annual support contract. Depending upon your needs and budget, you may choose any one of these options or combine several into a custom program suitable to your requirements.

When contacting Technical Support, include or be prepared to provide the following information:

- ▶ Product name and version number
- ▶ Product patch level (use `SHOW VERSION` command).
- ▶ Operating system serial number (displayed on bootup or use `SHOW SERIAL` command).
- ▶ Operating system version number (displayed on bootup or use `SHOW VERSION` command).
- ▶ Type of hardware being used.
- ▶ What happened and what you were doing when the problem occurred.
- ▶ The exact wording of any messages that appeared on the screen(s).
- ▶ How you tried to solve the problem.

Dealers and THEOS resellers may contact THEOS Technical Support by mail, fax, telephone or the Internet.

Contacting THEOS

Support

Mail: THEOS Technical Support
THEOS Software Corporation
1801 Oakland Boulevard, Suite 315
Walnut Creek, CA 94596-7000

Fax: 925-935-1177

Telephone: 925-935-1118 ext 211
Hours: Monday-Friday,
8:30am - 4:30pm, Pacific Time

Internet Email: support@theos-software.com

Internet WWW: <http://www.theos-software.com>

B: THEO+Server Support Files

The following files are distributed with the THEO+Server product:

Library	Member	Description
SYSTEM.MENU32	SETFTPD	Internal menus, literal text messages and context-sensitive help text for the Setup Server, FTP command.
	SETHTPD	Internal menus, literal text messages and context-sensitive help text for the Setup Server, HTTP command.
	SETSERVE	Internal menus, literal text messages and context-sensitive help text for the Setup Server command.
SYSTEM.TEOS32	BROWSCAP	The browser capabilities database. This information is used to map the browser's identification string to the capabilities of the browser.
	FTPDCFG	The configuration file maintained by the Setup Server, FTP program and used by the FTP server. This file is not distributed, but is created the first time that Setup Server, FTP is used.
	FTPSERVE	The THEOS FTP Server program.
	HTTPAUTH	File containing the user authorization specifications.
	HTTPDCFG	The configuration file maintained by the Setup Server, HTTP program and used by the HTTP server. This file is not distributed, but is created the first time that Setup Server, HTTP is used.
	SETFTPD	The Setup Server, FTP program.
	SETHTPD	The Setup Server, HTTP program.
	SETSERVE	The Setup Server program.
	WEBSERVE	The THEOS HTTP Server program.
/HTTPICON/	*	Collection of images and active server pages used for displaying directory browse views and server error messages.
† The “D” in FTPD and HTTPD refers to “Daemon.” The word “daemon” comes from Greek mythology where the daemons acted as messengers between the people and the gods.		

The following files are installed with the CGI ToolKit:

Library	Member	Description
/CGI/DATA	CUSTOMER.MASTER	Customer database.
	ORDERS.DETAIL	Orders line-item detail database.
	ORDERS.MASTER	Orders summary database.
/CGI/DATA/RESOURCE	STATE.CODES	State and Canadian province codes database.
/CGI/SOURCE	ADDRESS.BASIC	Common include file for CGI generated documents.
	CUST1.BASIC	Program generating customer and orders view.
	CUSTBRWS.BASIC	Program generating customer browse form.
	CUSTOMER.BASIC	Program generating customer entry form.
	MAINMENU.BASIC	Dispatch program from mainmenu.stm document.
	MAKEFILE.	Make file for compiling programs in example.
/WWW/EXAMPLES/CGI	DEFAULT.SSI	Main menu document.
/WWW/EXAMPLES/HTML	*	Various example documents showing usage of some HTML code and/or SSI directives.
/	CGI.BASIC	Include file declaring C language function protocol and defining the BASIC language functions in the CGI API.
SYSTEM.B3220LIB	CGI*	The C language functions in the CGI API.
SYSTEM.CMD32	CGITEST	Application used for testing a CGI applications.

C: FTP Log File

The FTP log file is created when the log file directory is defined (see “[Log Options](#)” on page 32) and the [Log Frequency](#) is not 0 (see page 33).

The name of the FTP log file is determined by the [Log Frequency](#) setting and possibly the date that the log file was created. A [Log Frequency](#) of 2, 3 or 4 cause the name of the FTP log file to change as it is rotated by day, week or month. The names used for the FTP log file are:

Code	Rotated	HTTP Log File Name
1	No	/logs/FTP.LOG
2	Daily	/logs/FTyymmdd.LOG
3	Weekly	/logs/FTyymmdd.LOG
4	Monthly	/logs/FTyymm01.LOG
<i>/logs/</i> is defined by the Log Directory field in Log Options menu. <i>yyyy</i> is the year number. <i>mm</i> is the month number. <i>dd</i> is the day number. For code 3, the day number is always the day number of the Monday of the week that the log file was started.		

The current log file is kept open by the FTP server while it is started.

The format for each record in the file is:

YYYY/MM/DD HH:MM:SS [N] (nnnnn) Message text

└── User ID number

└── Message type code

└── Date and time of event

The user ID number is a sequentially assigned user number. This number is assigned when a connection to the server is attempted. All messages relating to the connection will use the same ID number.

The message type code identifies the event type:

1. System events such as starting and stopping the server, server error conditions, *etc.*
2. FTP client commands.
3. GET file transfers.
4. PUT file transfers.
5. Connection, disconnection and security events such as invalid account or password.
6. FTP server replies to client requests and other messages sent to the client. Directory listing output is not logged.

D: HTTP Log File

The HTTP log file is created when the log file directory is defined (see “[Directory Setup](#)” on page 48) and the [Log File Frequency](#) is not 0 (see page 47).

The name of the HTTP log file is determined by the [Log File Frequency](#) setting and possibly the date that the log file was created. A [Log File Frequency](#) of 2, 3 or 4 cause the name of the HTTP log file to change as it is rotated by day, week or month. The names used for the HTTP log file are:

Code	Rotated	HTTP Log File Name
1	No	/logs/HTTP.LOG
2	Daily	/logs/HTyymmdd.LOG
3	Weekly	/logs/HTyymmdd.LOG
4	Monthly	/logs/HTymm01.LOG
<p>/logs/ is defined by the Logfile field in Directory Setup menu.</p> <p>yyyy is the year number.</p> <p>mm is the month number.</p> <p>dd is the day number. For code 3, the day number is always the day number of the Monday of the week that the log file was started.</p>		

The current log file is kept open by the HTTP server while it is started.

RequestingIP	-	-	[DateTime]	"Request"	Result	Size	"URL"	"UserAgent"
								Browser ID
								Referring document
								Size of document
								Code for request result**
								Request type and document name
								Date and time of request *
								Authenticated user name or "-"
								Server name or "-"
								IP address of User Agent making request

“[dd/mm/yyyy:hh:mm:ss -hhmm]”

**** The status codes are:**

Code	Meaning
200	Okay
301	Document moved permanently
302	Document moved temporarily
304	Not modified
400	Invalid request
401	Unauthorized access
404	Document not found
500	Internal server error
501	Not implemented
503	Service unavailable
505	HTTP version not supported

E: FTP Message Variables

The following variables may be used in any of the message text files used by the FTP server.

Variable	Meaning
%bdown	Number of bytes downloaded during this session
%btot	Number of bytes uploaded and downloaded during this session
%bup	Number of bytes uploaded during this session
%date	Today's date in MM/DD/YY format
%dir	Virtual current working directory
%fdown	Number of files downloaded during this session
%ftot	Number of files uploaded and downloaded during this session
%fup	Number of files uploaded during this session
%help	E-mail value defined in general options configuration
%host	Host name
%ip	User IP number or DNS name
%ipname	DNS name of the client
%ipnum	IP number of the client
%maxanonymous	Maximum number of anonymous users allowed
%maxusers	Maximum number of users allowed
%name	User name
%tconm	Connect time for this session, in minutes
%tcons	Connect time for this session, in seconds
%time	Local time in HH:MM AM/PM format
%u24h	Total number of users that have connected today
%uall	Total number of users that have connected to this server
%unow	Current user count

F: SSI Directives

Server Side Includes (SSI) are directives in a web document that are processed by the server before the document is given to the user agent. For the most part, these directives are replaced with text from another source creating a dynamic document that can be different each time that it is retrieved.

■ SSI Syntax

All SSI directives use the following, common syntax:

```
<!--#directive-name mode="value"-->
```

Notice that the delimiters for the SSI directive are the same delimiters used for HTML comments (a beginning `<!--` and a closing `-->`). When a page containing SSI directives is passed to a browser client without pre-processing the directives, the browser treats the directives as comments and does not display them.

Each of the SSI directive names start with the `#` character. Within the directive, only white space may separate the opening comment delimiter and the start of the directive name and between the end of the directive and the closing comment delimiter. For instance,

```
<!--#include file="common1.htm"-->
<!--  #include file="common2.htm"  -->
```

are valid, while

```
<!--some text #include file="common.htm" -->
```

is not valid because there is text characters between `<!--` and `#`.

The keywords used in SSI directives are case-insensitive. Thus, `#config`, `#Config` and `#CONFIG` are all valid and refer to the same directive.

■ SSI Directives

There are eight SSI directives supported by the THEOS HTTP Server:

<code>#config</code>	Sets the format for dates and times and file size or sets the message generated by an error in an SSI directive.
<code>#counter</code>	Sets the value of a counter or includes the value of the counter as text in the HTML file.
<code>#echo</code>	Includes the value of a variable as text in the HTML file.
<code>#exec</code>	Executes a CGI script or program file and includes the output of that script or program file as text in the HTML file.
<code>#lastmod</code>	Includes a file's last change date as text in the HTML file.
<code>#size</code>	Includes the file size of a file as text in the HTML file.
<code>#include</code>	Copies another file into the HTML file.
<code>#nossi</code>	Terminates the preprocessing of the remainder of this file. Remaining text is passed on as-is.

#config

The #config directive specifies the message text used when an error is detected during SSI processing or the format used for file sizes, dates or counters.

```
#config CounterFmt="counter-format"

#config ErrMsg="text"

#config SizeFmt="size-format"

#config TimeFmt="time-format"
```

<i>counter-format</i>	»	format string for counters
<i>size-format</i>	»	format string for file sizes
<i>text</i>	»	SSI error message text
<i>time-format</i>	»	format string for dates and times

Operation: Unlike the other SSI directives, the #config directive does not put any text into the web page. Instead, it defines the format of the text inserted by other directives.

When a web page is processed, the error message and formats are initialized to their default settings. The settings specified by this directive are in effect until another #config directive is processed in this file or one of its included files.

CounterFmt= Specifies the format used to display counters reported by the #counter directive.

```
#Config CounterFmt="format"
```

ErrMsg= Indicates that any errors detected by the SSI preprocessor are to be reported with a user-specified text message.

```
#Config ErrMsg="text"
```

SizeFmt= Defines the format used for file size reports generated by the `#fsize` directive. There are two formats available for file sizes.

#Config SizeFmt="Abbrev"

Specifies that the `#fsize` reports file sizes as a count of kilobytes. This is the default format for file sizes.

#Config SizeFmt="Bytes"

Specifies that the `#fsize` reports file sizes as a count of bytes.

TimeFmt= Specifies the format used to display dates and times reported by the `#echo` and `#flastmod` directives.

#Config TimeFmt="format"

The *format* string may contain literal characters and format specifications identical to the specifications used in the C language `strftime` function.

Returns: No text or value is returned by these directives.

Notes: **CounterFmt=** The *format* string may contain literal characters and format specifications identical to the specifications used in the C language `printf` function. The default format for counters is `"%d"` which displays counters as a standard integer with no thousands separators.

Some common counter formats that you might use would be `"%,d"` for comma-separated values and `"%.5d"` for zero-padded, five-digit displays.

ErrMsg= When this directive is not used in a file, the default error messages are used:

```
Unrecognized SSI #command-text
Invalid #config mode="setting"
Invalid #counter mode="name"
Cannot #echo var="name"
Cannot #exec cmd="command-name"
Cannot #flastmod mode="filename"
Cannot #fsize mode="filename"
Cannot #include mode="filename"
```

Since you cannot specify any variables in the user-specified error message text, it will not contain any useful details about the error.

The error message text cannot be a null string.

SizeFmt= The default format for file sizes is “Abbrev.”

TimeFmt= The default format for dates and times is “%A %B %d %Y” corresponding to:

weekday-name space month-name space day-number space year-number

as in “Thursday March 05 1998.”

The format codes that may be used in the *format* string include:

Char	Time Element Substitution
%a	Abbreviated weekday name (Sun, Mon, <i>etc.</i>) ^{† A}
%A	Full weekday name (Sunday, Monday, <i>etc.</i>) ^{† A}
%b	Abbreviated month name (Jan, Feb, <i>etc.</i>) ^{† A}
%B	Full month name (January, February, <i>etc.</i>) ^{† A}
%c	Date and time in standard DATEFORM format. For instance, DATEFORM 1 produces: MM/DD/YYYY HH:MM:SS ^{‡ A}
%C	Date and time in current locale’s long format, using “%J %B %Y %H:%M” ^U
%d	Day number of month (01-31) ^A
%D	Date in format: MM/DD/YY ^U
%e	Day number of month (1-31) with single digits preceded by a space character ^U
%h	Abbreviated month name (Jan, Feb, <i>etc.</i>), synonym to “%b” ^U
[†] The SYSTEM.TEOS32.MESSAGE <i>n</i> file is used to get the actual names. [‡] The date format is determined by the DATEFORM environment variable. ^A Conforms to ANSI minimal specifications. ^U Extensions to ANSI added by UNIX system V.	

Table 1: #Config TimeFmt Codes

Char	Time Element Substitution
%H	Hour number, 24-hour clock (00-23) ^A
%I	Hour number, 12-hour clock (01-12) ^A
%j	Julian day number of year (001-366) ^A
%J	Day number of month (1-31) with single digits not preceded by a space or zero character
%k	Hour number (0-23) with single digits preceded by a space character ^U
%l	Hour number (1-12) with single digits preceded by a space character ^U
%m	Month number (01-12) ^A
%M	Minute number (00-59) ^A
%n	Same as a “\n” ^U
%p	AM or PM ^{† A}
%r	Time in standard 12-hour format, using “%I:%M:%S %p” ^U
%R	Time in standard 24-hour format, using “%H:%M” ^U
%S	Second number (00-59) ^A
%t	Same as a “\t” ^U
%T	Time in standard 24-hour format: HH:MM:SS ^U
%U	Week number of year, Sunday is 1 st day of week (00-52) ^A
%w	Weekday number (0=Sunday through 6=Saturday) ^A
%W	Week number of year, Monday is 1 st day of week (00-52) ^A
%x	Date in standard format: MM/DD/YYYY ^{‡ A}
%X	Time in standard 24-hour format: HH:MM:SS ^A
%y	Year number in century (00-99) ^A
%Y	Year number (1900-2106) ^A
[†] The SYSTEM.TEOS32.MESSAGES file is used to get the actual names. [‡] The date format is determined by the DATEFORM environment variable. ^A Conforms to ANSI minimal specifications. ^U Extensions to ANSI added by UNIX system V.	

Table 1: #Config TimeFmt Codes

Char	Time Element Substitution
%Z	Timezone name ^A
%%	Literal percent character ^A
[†] The SYSTEM.TEOS32.MESSAGE <i>n</i> file is used to get the actual names. [‡] The date format is determined by the DATEFORM environment variable. ^A Conforms to ANSI minimal specifications. ^U Extensions to ANSI added by UNIX system V.	

Table 1: #Config TimeFmt Codes

See also: [#counter](#), [#echo](#), [#flastmod](#), [#size](#)

#counter

This directive maintains and includes the value of a counter.

```
#counter Name="counter-name"
```

```
#counter NoDisp="counter-name"
```

```
#counter NoIncr="counter-name"
```

```
#counter Reset="counter-name"
```

```
#counter Set="counter-name=value"
```

counter-name » counter name

value » integer value to assign to counter

SSI

Operation: There are five forms of the #counter directive.

Name= Increments the counter and returns that incremented value. This is the common usage of the #counter directive.

```
#counter name="counter-name"
```

This directive should be used on a specific counter only one time in a page. If you want to use the current value of a counter a second time in a web page, use the noincr form for the secondary references.

NoDisp= Increments the counter but does not return the value. This form allows a web page to use a “hidden” counter.

```
#counter nodisp="counter-name"
```

NoIncr= Returns the current value of the counter. The counter is not incremented.

```
#counter noincr="counter-name"
```

Reset= Resets the value of the counter to zero.

```
#counter reset="counter-name"
```


Set= Sets the counter to a specific value.

#counter set="counter-name=value"

Returns: **Name=** Returns the incremented value of the counter.

NoDisp= Returns a null string.

NoIncr= Returns the current value of the counter.

Reset= Returns a null string.

Set= Returns a null string.

Notes: Counters are built-in usage counters available on the THEOS HTTP Server. With these built-in counters it is easy to count the number of times that a page is viewed, a file is downloaded, a link is followed or any other usage that you want that can be counted with the number of times that a web page is downloaded.

The **#counter name="name"** and the **#counter nodisp="name"** directives increment the counter while the other forms of the **#counter** directive either set the counter to a specific value or return the current value of the counter.

Counter names and values are maintained in the HTTP Server configuration file `SYSTEM.THEOS32.HTTPDCFG`.

An example usage of the **#counter** directive:

```
<center>
This site has been accessed <!--#counter name="site-count"--> times
since January 1, 1998.
</center>
```

If you have some procedure that resets some counters at the beginning of the month you could use:

```
<center>
This site has been accessed <!--#counter name="month-count"--> times
this month. <!--#counter name="total-count"--> users have accessed
the site since January 1, 1998.
</center>
```

Of course, you could combine usage of the `#counter` directive with some active server page code to make the display more intelligent:

```
<center>
This site has been accessed <!--#counter name="month-count"-->
<% If <!--#counter noincr="month-count"--> > 1 Then %>
time
<% Else %>
times
<% EndIf %>
this month. <!--#counter name="total-count"-->
<% If <!--#counter noincr="total-count"--> > 1 Then %>
users have
<% Else %>
user has
<% EndIf %>
accessed the site since January 1, 1998.
</center>
```

Note that the `NoIncr=` form of the `#counter` directive is used when testing the value of the counter. If the `Name=` form was used, the counter would be incremented twice!

The above example produces different text depending upon whether the counts are greater than one or not.

```
This site has been accessed 1 time this month. 1 user has accessed
the site since January 1, 1998.
```

or maybe:

```
This site has been accessed 2 times this month. 10 users have
accessed the site since January 1, 1998.
```

See also: [#config CounterFmt=](#)

#echo

Gets the value of a server or HTTP header variable.

#echo Var="server-variable"

#echo Var="HTTP_header-variable"

header-variable

»

HTTP header variable name (see Table 3: “HTTP Header Variables” on page 149)

server-variable

»

server variable name (see Table 2: “Server Variables” on page 147)

- Operation:** The *header-variable* or *server-variable* name is evaluated with the value of that variable returned.
- Returns:** The value of the referenced variable is returned and replaces the SSI directive.
- Notes:** The *header-variable* and *server-variable* names supported by the THEOS HTTP Server include:

Variable-name	Meaning
AUTH_TYPE	This is the protocol-specific authentication method used to validate the user. For the THEOS HTTP Server, “Basic” is always returned.
CONTENT_LENGTH	Length of POST document
CONTENT_TYPE	GET or POST
DOCUMENT	The path and file name of the current web page, relative to the server.
DOCUMENT_URI	The path and file name of the current web page, relative to the server home directory.

Table 2: Server Variables

Variable-name	Meaning
DATE_GMT	The date and time on the server adjusted to Greenwich Mean Time (GTM) which is now called UTC time. This date/time is formatted according to the current <code>#config TimeFmt=</code> setting.
DATE_LOCAL	The date and time on the server in its local time. This date/time is formatted according to the current <code>#config TimeFmt=</code> setting.
GATEWAY_INTERFACE	The revision of the CGI specification to which this server complies. Format: CGI/revision
LAST_MODIFIED	The date and time that the current web page file was last changed. This date/time is formatted according to the current <code>#config TimeFmt=</code> setting.
PATH_INFO	This variable is always blank except when a CGI application is executed.
PATH_TRANSLATED	This variable is always blank except when a CGI application is executed.
QUERY_STRING	If this web page was referenced with a query string specified, this variable contains the text following the “?”. It is not decoded in any fashion.
QUERY_STRING_UNESCAPED	The query string with special characters not escaped.
REMOTE_ADDR	The IP address of the remote client requesting this web page. (When a proxy server is used between the server and the client, this may be the proxy server IP address.)
REMOTE_HOST	The host name of the remote client requesting this web page. If the server does not have this information, it uses the value of REMOTE_ADDR. (When a proxy server is used between the server and the client, this may be the proxy server host name.)

Table 2: Server Variables

Variable-name	Meaning
REQUEST_METHOD	Contains the method used to invoke this document. It is either “GET” or “POST.”
SCRIPT_NAME	The virtual path to this web page being executed.
SERVER_NAME	The server’s hostname, as defined in the SYSTEM.TEOS32.HTTPDCFG file.
SERVER_PORT	The port number used by the server.
SERVER_PROTOCOL	The name and revision of the information protocol used by the server. Format: protocol/revision, i.e., HTTP/1.0.
SERVER_SOFTWARE	The name and version of the HTTP server software. Format: name/version, i.e., THEO+Server/1.0.

Table 2: Server Variables

Header Variable	Meaning
ACCEPT	The list of the MIME types accepted by the browser client.
ACCEPT_LANGUAGE	The list of the human languages accepted by the browser client.
COOKIE	The value of the cookie sent by the browser client.
REFERER	The URL of the web page that the browser client used to link to this document.
USER_AGENT	The name of the browser client software. For instance, “Mozilla/4.0 (compatible; MSIE 4.0; Windows 95)” or “Mozilla/4.03 [en] (Win95; I ;Nav).”

Table 3: HTTP Header Variables

#exec

The **#exec** directive executes a command or CGI program and captures the output of that program (stdout) as the text to be inserted into the HTML file.

```
#exec CGI="cgi-command-line"
```

```
#exec CMD="command-line"
```

cgi-command-line » path and name of CGI program and parameters

command-line » path and name of command and command-line options

Operation: **CGI=** This directive invokes a compiled CGI script program.

```
#Exec CGI="cginame?name=value"
```

CMD= This directive invokes a compiled program.

```
#Exec CMD="command-line"
```

The *command-line* is the path and name of the program, optionally followed by any command-line arguments used by the program.

Returns: All of the CGI's output to stdout or the command's output to stdout is included in this web page at this location in the file.

Notes: **CGI=** The *CGI-command-line* is the path and name of the CGI script program optionally followed by a question mark character (?) and the query string parameters used by the script program.

For instance:

```
#exec cgi="/cgi/cgiscript?directory=/www/test/"
```

When a query string is used, it must be "URL encoded." That is, space characters are replaced by the plus-sign character (+), letters, digits, "\$," "-", "_" and period characters are unchanged, and other characters are converted to "%hh" where *hh* is the hexadecimal value of the character.

CMD= Any output by the program to stdout is captured and inserted into this document in place of this directive.

For instance:

```
<pre><!--#exec cmd="list some.file (nohead numbered"-->
</pre>
```

Because this directive executes a program on your server, it can be a security and data-integrity concern.

See also: [#include](#)

#lastmod

This directive gets a file's last change date and time and includes it in the resulting HTML file.

```
#lastmod File="filename"
```

```
#lastmod Virtual="filename"
```

filename » path and name of file

Operation: The last change date for *filename* is determined and returned.

File= In this mode of the #lastmod directive, the *filename* always refers to the name and path of a file relative to the current document's path. That is, it is relative to this file. The current document's path is prepended onto the *filename*.

For instance, if the current document is /pages/examples/testpage.htm

Reference	File Referenced
File="page2.htm"	/pages/examples/page2.htm
File=".. /page2.htm"	/pages/page2.htm

Virtual= In this mode of the #lastmod directive, the *filename* always refers to the name and path of a file relative to the virtual root of the HTTP server.

For instance, if the current document is /pages/examples/testpage.htm

Reference	File Referenced
Virtual="page2.htm"	/page2.htm
Virtual="/Resource/page2.htm"	/Resource/page2.htm

Using a filename specification that starts with "/CGI/" will be interpreted as a reference to the directory specified in the HTTP server's configuration for the CGI directive.

Returns: The last change date and time.

Notes: The date and time is formatted according to the current [#config TimeFmt=](#) setting.

File references are always to files that are within the scope of the HTTP server's virtual directories. A file outside of this scope cannot be referenced.

See also: [#config TimeFmt=](#)

#fsize

This directive gets a file's size and includes it in the resulting HTML file.

```
#fsize File="filename"
```

```
#fsize Virtual="filename"
```

filename » path and name of file

Operation: The size of *filename* is determined and returned.

The File= and Virtual= forms of this directive operate the same as those forms of the [#lastmod](#) directive described on page [152](#).

Returns: The size of the file in number of kilobytes or bytes, depending upon the current [#config SizeFmt=](#) setting.

Notes: The size is formatted according to the current [#config SizeFmt=](#) setting. If no [#config SizeFmt=](#) directive is in effect, the default size format is “Abbrev” which formats the size as a count of the number of kilobytes.

See also: [#config SizeFmt=](#)

#include

This directive gets the text from another file and copies it into the file being processed.

```
#include File="filename"
```

```
#include Virtual="filename"
```

filename » path and name of file

Operation: The referenced file is located and its contents are included in this web page.

The File= and Virtual= forms of this directive operate the same as those forms of the [#flastmod](#) directive described on page [152](#).

Returns: The contents of the referenced file.

Notes: After the contents of the referenced file are included in the current web page, it participates in SSI preprocessing. This means that the included text may include SSI directives, including other `#include` directives.

See also: [#exec](#)

#nossi

This directive terminates SSI preprocessing for this web page.

```
#nossi
```

Operation: SSI preprocessing for this web page is terminated. Any remaining text in the file is included in the web page without analysis.

Returns: No value or text is returned by this directive.

Notes: By using the #nossi directive in a large web page after all of the directives have been processed, the remainder of the web page text can be copied to the web page without analysis. Depending upon the size of the remaining document, this can reduce the time required to download a page.

Another common usage of this directive is for sites that enable SSI preprocessing for all file types. Any web pages that do not use SSI directives can specify this directive at the start of the file.

If there are more SSI directives in the file after this #nossi directive, they are included in the resulting web page but remain as comments.

G: MultiUser Basic Language CGI API Functions

The functions described in this appendix are included with THEO+Server and are installed if you choose the “Install CGI ToolKit” menu item during installation. They are installed in the /SYSTEM.B3220LIB directory or are defined within the CGI.BASIC file installed in the root directory.

The CGI Application Program Interface (API) for MultiUser BASIC Version 2.0 provides an interface that programs can use to create CGI script programs invoked by HTML web page FORMs, SSI [#exec](#) directives or other CGI script programs.

Refer to “[Common Gateway Interface \(CGI\)](#)” on page 113 for a description of the Common Gateway Interface and how it is used by the THEOS HTTP Server. Also refer to Appendix I: “[CGITEST Command](#)” starting on page 237. That command can assist you in testing your CGI applications.

■ CGI Functions

The MultiUser BASIC language CGI API includes the following functions declared in the files CGI.BASIC and CGIJAVA.BASIC. Any program using these functions must include these files at the beginning of each program because it declares that the functions use the 2.0 style interface and it declares the formal calling sequence for the functions.

You only need to include the CGIJAVA.BASIC file if you actually use the functions defined in it.

There is no interpreter interface definition provided because CGI programs can only be invoked as compiled programs.

```
DECLARE CALL CGI.INIT
DECLARE CALL CGI.BEGIN ( title$, txtcolor$, bgcolor$ )
DECLARE CALL CGI.FINISH
DECLARE CALL CGI.ERROR ( text$ )
DECLARE CALL CGI.LOCATION ( url$ )
DECLARE CALL CGI.STATUS ( status%, desc$ )
DECLARE CALL CGI.PUT.COOKIE ( name$, value$, expire, path$, domain$ )
FN.CGI.BROWSER$ ( capability$ )
FN.CGI.GET$ ( name$ )
FN.CGI.GET.COOKIE$ ( name$ )
FN.CGI.GET.FORM$ ( name$ )
FN.FORM$ ( program$ )
```

```

FN.FORM.BUTTON$ ( name$, label$ )
FN.FORM.CHECKBOX$ ( name$, checked%, value$, text$ )
FN.FORM.HIDDEN$ ( name$, value$ )
FN.FORM.INPUT$ ( name$, size%, maxlength%, value$ )
FN.FORM.PASSWORD$ ( name$, size%, maxlength%, value$ )
FN.FORM.RADIO$ ( name$, checked%, value$, text$ )
FN.FORM.RESET$ ( label$ )
FN.FORM.SELECT$ ( name$, size% )
FN.FORM.SELECT.OPTION$ ( value$, selected%, text$ )
FN.FORM.SUBMIT$ ( label$ )
FN.FORM.TEXTAREA$ ( name$, size%, rows%, text$ )
FN.HTML.BREAK$
FN.HTML.CENTER$ ( text$ )
FN.HTML.DECODE$ ( text$ )
FN.HTML.ENCODER$ ( text$ )
FN.HTML.HEADING$ ( number%, text$ )
FN.HTML.IMAGE$ ( url$, iwidth%, iheight%, border%, alt$ )
FN.HTML.LINK$ ( text$, url$ )
FN.HTML.LIST.ITEM$
FN.HTML.MAILTO$ ( text$, url$ )
FN.HTML.PARA$
FN.HTML.RULE$ ( width% )
FN.HTML.TAG$ ( tag$, OnOff% )
FN.JAVA.FORM$ ( program$ name$, submit$ reset$ )
FN.JAVA.BUTTON$ ( name$, label$, focus$, blur$, change$, mouseover$, mouseout
$ )
FN.JAVA.CHECKBOX$ ( name$, checked%, value$, text$, focus$, blur$, change$,
mouseover$, mouseout$ )
FN.JAVA.INPUT$ ( name$, size%, maxlength%, value$, focus$, blur$, change$,
mouseover$, mouseout$ )
FN.JAVA.PASSWORD$ ( name$, size%, maxlength%, value$, focus$, blur$, change$,
mouseover$, mouseout$ )
FN.JAVA.RADIO$ ( name$, checked%, value$, text$, focus$, blur$, change$,
mouseover$, mouseout$ )
FN.JAVA.RESET$ ( label$, focus$, blur$, change$, mouseover$, mouseout$ )
FN.JAVA.SELECT$ ( name$, size%, focus$, blur$, change$ )
FN.JAVA.SUBMIT$ ( label$, focus$, blur$, change$, mouseover$, mouseout$ )
FN.JAVA.TEXTAREA$ ( name$, size%, rows%, text$, focus$, blur$, change$,
mouseover$, mouseout$ )
FN.TABLE$ ( begin%, border% )
FN.TABLE.DATA$ ( text$ )
FN.TABLE.HEAD$ ( text$ )
FN.TABLE.RECORD$ ( text$ )
FN.URL.DECODE$ ( text$ )
FN.URL.ENCODER$ ( text$ )

```

CGI.INIT, CGI.BEGIN, CGI.FINISH, CGI.PUT.COOKIE

These functions are used to initialize the interface between the calling environment and this CGI program and to create the required headers for beginning and ending a web page.

```
CALL CGI.BEGIN ( title$, txtcolor$, bgcolor$ )
```

```
CALL CGI.FINISH
```

```
CALL CGI.INIT
```

```
CALL CGI.PUT.COOKIE ( name$, value$, expire, path$, domain$ )
```

<i>bgcolor\$</i>	»	Color name or code for web page background
<i>domain\$</i>	»	Text string containing domain name of this web site
<i>expdate\$</i>	»	Expiration date for cookie
<i>name\$</i>	»	Cookie field name
<i>path\$</i>	»	Path within domain that cookie applies to
<i>title\$</i>	»	Text string for the web page title
<i>txtcolor\$</i>	»	Color name or code for text
<i>value\$</i>	»	Value for cookie field

Operation: **CGI.BEGIN** Outputs the text lines that start a web page, defining the page title and the body text colors. This function is also responsible for generating the text lines that define all of the cookies that have been specified with the [CGI.PUT.COOKIE](#) function.

CGI.FINISH Outputs the text lines that terminate the web page body section and the web page itself. After outputting these lines, the program exits, closing all files.

CGI.INIT Using the FORM method, the variable names and values passed to this program are extracted and saved for subsequent access by the [FN.CGI.GET\\$](#) and [FN.CGI.GET.FORM\\$](#) functions.

CGI.PUT.COOKIE Adds a cookie field definition to the web page being generated. Cookies can only be defined by a web document when no HTML code has been generated. In other words, after a CALL to [CGI.INIT](#) and before a CALL to [CGI.BEGIN](#).

The **CGI.BEGIN** function must be used because it is the routine that actually outputs the cookie definition to **stdout**.

The *name\$* and *value\$* fields refer to the cookie name and the new value for the cookie. *domain\$* and *path\$* refer to the web documents that will return this cookie value in the future. (See [Notes](#): section for more information about these parameters.)

The *expires\$* field specifies how long this cookie value is valid and is specified with a day number such as is created by the **DAY** function. For instance, to have a cookie expire tomorrow use a specification of **DAY (DATE\$ (0)) +1**. This will encode it as expiring at midnight tonight.

Returns: The **CGI.BEGIN** and **CGI.FINISH** functions do not return a value and they do not set the **CALL.RETURN.VALUE**. Instead, they print text to the **stdout** device.

CGI.INIT The **CALL.RETURN.VALUE** function is set to a success/fail indicator: A zero indicates success, nonzero indicates an error.

CGI.PUT.COOKIE When *name\$* is an empty string, no cookie definition string is generated. When it is not an empty string, the cookie definition is created but not output until the **CGI.BEGIN** function is used.

Errors: **CGI.BEGIN** If **CGI.INIT** is not used prior to calling this function, no headers are generated.

CGI.INIT A nonzero **CALL.RETURN.VALUE** indicates an error. Possible values include:

Value	Meaning
0	Success.
1	Form method is POST but no content was specified.
2	Form method is GET but not content was specified.
3	Form method invalid.

Notes:

CGI.BEGIN The text and background colors may be specified with their hexadecimal values or with one of the standard color names.

The standard color names that can be used include:

Aqua	Gray	Navy	Silver
Black	Green	Olive	Teal
Blue	Lime	Purple	White
Fuchsia	Maroon	Red	Yellow

Table 4: HTML Standard Color Names

Hexadecimal color values are specified with a leading hash-mark character (#) followed by three or six hexadecimal digit characters. For instance, #C0C0C0 is the code for “Silver.” A three-hexadecimal-digit color number is expanded into a six-hexadecimal-digit color number by replicating each of the three digits. For instance, #ABC is the same as #AABBCC.

The lines of text generated by this function are:

```
Content-Type: text/html
```

```
any cookie definitions here...
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
```

```
<HTML>
```

```
<HEAD>
```

```
<TITLE>title$</TITLE>
```

```
</HEAD>
```

```
<BODY TEXT="txtcolor$" BGCOLOR="bgcolor$">
```

When *title\$* is a null string, the HEAD section is omitted.

CGI.FINISH The lines of text generated are:

```
</BODY>
```

```
</HTML>
```

CGI.PUT.COOKIE Cookies are name/value pairs that are maintained by some browsers on the client's system. These cookie name/value pairs are delivered by the browser when it requests a web document from a server. Not all browsers support cookies and for those that do, the user may disable them.

The cookie is delivered to the server only if the server's domain and the path of the web document requesting it match the *path\$* and *domain\$* parameters defined for the cookie.

The *domain\$* specifies a server domain. For instance, specifying a *domain\$* of "www.theos-software.com" indicates that the cookie is associated with requests to the web server at THEOS Software Corporation. A request to a server at another site or even another server at THEOS will not send the cookie.

You can specify that a cookie is associated with all of the server machines as a site by omitting the machine name in the *domain\$* value. For instance, to specify that all servers at THEOS are to receive the cookie, specify a *domain\$* of ".theos-software.com".

The *path\$* value is used to restrict a cookie to a specific web document or groups of documents within *domain\$*. To restrict a cookie to a single web document use a *path\$* value of the URL of the document (minus the domain specification). For instance: "/customer/custstat.htm".

You can indicate that a cookie is associated with a group of web documents by using a *path\$* that does not include a specific document name. For instance a *path\$* of "/customer" indicates that all web documents in the customer directory tree will receive this cookie value.

It is best to specify a *path\$* value of "/" because of the ambiguity of the path for a CGI application and because of errors in some implementations of various browsers.

Restrictions: **CGI.BEGIN** The **CGI.INIT** function must be called prior to this function call.

CGI.INIT This function must be used prior to calling the **CGI.BEGIN** function and prior to using the **FN.CGI.GET\$** or **FN.CGI.GET.FORM\$** functions.

The storage area used for saving the variable names and values passed to this program has large, but limited space available to it. Specifically, the total length of all variables names is 512 characters and the total length of all variable values is 4,000 characters.

Many of the other FN functions in this API make use of the subroutines defined within this function. The **CGI.INIT** function

must be called in the program to make these subroutines available to those other functions.

CGI.PUT.COOKIE This function is only effective when it is called prior to using the [CGI.BEGIN](#) function. It creates MIME headers that must be defined prior to defining other MIME headers or HTML lines. Specifically, the CALL [CGI.PUT.COOKIE](#) is used after a CALL [CGI.INIT](#) but before a CALL [CGI.BEGIN](#).

See also: [FN.CGI.GET\\$](#), [FN.CGI.GET.COOKIE\\$](#), [FN.CGI.GET.FORM\\$](#)

CGI.ERROR, CGI.LOCATION, CGI.STATUS

These functions generate an error message, redirected page message or a status message and exit the program.

CALL CGI.ERROR (*text*\$)

CALL CGI.LOCATION (*url*\$)

CALL CGI.STATUS (*status*%, *desc*\$)

<i>desc</i> \$	»	Description text for status code
<i>status</i> %	»	Status code to display
<i>text</i> \$	»	Error message text
<i>url</i> \$	»	<u>U</u> niform <u>R</u> esource <u>L</u> ocator for redirected page

Operation: **CGI.ERROR** Outputs *text*\$ as an error message in the web page followed by the text lines that terminate the web page body section and the web page itself. After outputting these lines, the program exits, closing all files.

CGI.LOCATION Outputs a redirection message and exits the program.

CGI.STATUS Outputs a status message using the *status*% code and the *desc*\$ text for that status code.

Returns: These functions do not return any value because they will exit the program without returning control to your program.

Errors: No errors are detected or reported by these functions.

Notes:

CGI.ERROR If the [CGI.BEGIN](#) function has not been used prior to this function the following lines are generated:

```
Content-Type: text/html
```

```
<HTML>
<BODY TEXT="#000000" BGCOLOR="#FFFFFF">
```

The following lines are always generated:

```
<H1>text$</H1>
</BODY>
</HTML>
```

Notice that the message text is displayed as a level-one header.

CGI.LOCATION Creates the MIME header for document redirection:

```
Location: url$
```

When specifying the *url\$*, be sure to use a full path and name specification.

CGI.STATUS The line of text generated is:

```
Status: status% desc$
```

Valid *status%* code values are:

Code	Meaning
200	Okay
301	Document moved permanently
302	Document moved temporarily
304	Not modified
400	Invalid request
401	Unauthorized access
404	Document not found
500	Internal server error
501	Not implemented
503	Service unavailable
505	HTTP version not supported

See also:

[CGI.BEGIN](#)

FN.CGI.BROWSER\$

This function reports on the HTTP client browser's capability, if known to the server.

FN.CGI.BROWSER\$ (*capability\$*)

capability\$ » Name of the capability to report on

Operation: The current browser definition is found in the browser capabilities file (SYSTEM.TEOS32.BROWSCAP) and the requested *capability\$* is returned.

The browser definition is defined in the environment variable HTTP_USER_AGENT, which is set each time that a browser requests a page.

Returns: The corresponding value of *capability\$* in the browser's capability entry in the browser capabilities file. A null string is returned if the browser is unknown, the browser capabilities file cannot be found, or if the capability is undefined for that browser.

Errors: No errors are reported by this functions.

Notes: The definition for the various client browsers varies from browser to browser and within the various versions and configurations of a particular browser. Some of the common names for the capabilities of a browser are:

Capability	Meaning
Browser	Name of browser
Platform	Operating system name
Version	Version of browser, for instance, 4.01
Frames	True or False value indicating that browser supports frames.
Tables	True or False value indicating that browser supports tables.
Cookies	True or False value indicating that browser supports cookies.

Table 5: Browser Capabilities

Capability	Meaning
Background Sounds	True or False value indicating that browser supports background sounds.
VBScript	True or False value indicating that browser supports Visual Basic Scripts.
JavaScript	True or False value indicating that browser supports Java Scripts.
TextOnly	True or False value indicating that the browser does not support images or other graphics.

Table 5: Browser Capabilities

Restrictions: The [CGI.INIT](#) function must be used in your program because it defines sub-functions used by this function.

The file `SYSTEM.THEOS32.BROWSCAP` must exist on the server's system.

FN.CGI.GET functions

These functions get the variable values that were passed to the CGI program from the FORM used to invoke the program or from the cookie associated with the program/site.

FN.CGI.GET\$ (*name\$*)

FN.CGI.GET.COOKIE\$ (*name\$*)

FN.CGI.GET.FORM\$ (*name\$*)

name\$ » Parameter or cookie variable name

Operation: **FN.CGI.GET\$** Gets the value of a FORM parameter variable or cookie variable with a variable name of *name\$*. FORMs variables are searched first and then, if no matching forms variable is found, cookie variables are searched.

FN.CGI.GET.COOKIE\$ Gets the value of the cookie variable *name\$*.

FN.CGI.GET.FORM\$ Gets the value of a FORM parameter variable *name\$*.

Returns: The value of the FORM or cookie variable with a name of *name\$*. If *name\$* cannot be found a null string is returned.

Errors: No errors are reported by these functions. If the *name\$* does not exist in the parameters or cookie, a null string is returned.

Notes: The value of *name\$* is case-insensitive.

FN.CGI.GET\$ Uses the [FN.CGI.GET.COOKIE\\$](#) and [FN.CGI.GET.FORM\\$](#) functions.

FN.CGI.GET.FORM\$ When the value of a parameter variable starts with the hash character (#), the parameter is treated as a synonym definition. The remainder of the value is treated as the actual name of the variable. These synonym definitions can be recursive. For instance, if the variable list passed by a FORM is:

SSN=123-45-6789
CUSTID=#SSN
ID=#CUSTID

Then a `FN.CGI.GET.FORM$ ("ID")` will return "123-45-6789."

FN.CGI.GET.COOKIE\$ Cookies are name/value pairs that are maintained by some browsers on the client's system. These cookie name/value pairs are delivered by the browser when it requests a web document from a server. Not all browsers support cookies and for those that do, the user may disable them.

Restrictions: The length of *name\$* must be less than 100 characters.

FN.CGI.GET.FORM\$ The [CGI.INIT](#) function converts the parameter list from the calling environment into a format that is usable by this CGI function. It must be used prior to using any of these functions.

See also: [CGI.INIT](#), [CGI.PUT.COOKIE](#)

FN.FORM functions

This set of functions generate text lines that define a FORM and its input fields.

CGI

FN.FORM\$ (*program*\$)

FN.FORM.BUTTON\$ (*name*\$, *label*\$)

FN.FORM.CHECKBOX\$ (*name*\$, *checked*%, *value*\$, *text*\$)

FN.FORM.HIDDEN\$ (*name*\$, *value*\$)

FN.FORM.INPUT\$ (*name*\$, *size*%, *maxlength*%, *value*\$)

FN.FORM.PASSWORD\$ (*name*\$, *size*%, *maxlength*%, *value*\$)

FN.FORM.RADIO\$ (*name*\$, *checked*%, *value*\$, *text*\$)

FN.FORM.RESET\$ (*label*\$)

FN.FORM.SELECT\$ (*name*\$, *size*%)

FN.FORM.SELECT.OPTION\$ (*value*\$, *selected*%, *text*\$)

FN.FORM.SUBMIT\$ (*label*\$)

FN.FORM.TEXTAREA\$ (*name*\$, *size*%, *rows*%, *text*\$)

<i>checked</i> %	»	Boolean indicator for checkbox or radio button
<i>text</i> \$	»	Label text for reset or submit button
<i>maxlength</i> %	»	Maximum length for input text
<i>name</i> \$	»	Name of form field
<i>program</i> \$	»	Name of CGI program that will process FORM response
<i>rows</i> %	»	Height of multi-line input area
<i>selected</i> %	»	Boolean indicator for selection field
<i>size</i> %	»	Width of input or selection field
<i>text</i> \$	»	Label text defined after input field
<i>value</i> \$	»	Value string for input field

Operation: **FN.FORM\$** Begin or end a FORM definition. If *program\$* is not a null string, a new FORM definition is started with the ACTION term specified as *program\$* and the METHOD specified as POST.

When *program\$* is a null string, the FORM is terminated.

FN.FORM.BUTTON\$ Defines a FORM input field of TYPE="SUBMIT" with a VALUE="*label\$*" attribute. This form of a submit button uses the value of *label\$* for the label of the button and defines a name/value pair that is included when the button is used to submit the form.

This format of the submit button is used when multiple submit buttons are displayed. The name/value pair is included so that the destination program (see [FN.FORM\\$](#) function) can distinguish which button was used to submit the form values.

```
PRINT FN.FORM.BUTTON$("item","Use It")
```

displays as:

When selected it defines item="Use It".

FN.FORM.CHECKBOX\$ Defines a FORM input field of TYPE="CHECKBOX".

```
PRINT FN.FORM.CHECKBOX$("item",1,"item1","some text")
```

displays as: ☒ Some text

When selected it defines item="item1".

FN.FORM.HIDDEN\$ Defines a FORM input field of TYPE="HIDDEN". Nothing is displayed by this field but it does define a name/value pair.

FN.FORM.INPUT\$ Defines a FORM input field of TYPE="TEXT".

```
PRINT "some text ";FN.FORM.INPUT$("item",5,10,"item1")
```

displays as: Some text

When selected it defines item="text-entered".

FN.FORM.PASSWORD\$ Defines a FORM input field of TYPE="PASSWORD". A PASSWORD field is just like an INPUT field except that the value of the text in the field and any text typed by the user is displayed as asterisks instead of the actual characters.

```
PRINT "Your key";FN.FORM.PASSWORD$("key",10,10,"cUrReNt")
```

displays as: Your key

When selected it defines item="text-entered".

FN.FORM.RADIO\$ Defines a FORM input field of TYPE="RADIO".

```
PRINT FN.FORM.RADIO$("item",1,"item1","some text")
```

displays as: ☒ Some text

When selected it defines item="item1".

FN.FORM.RESET\$ Defines a FORM input field of TYPE="RESET".

```
PRINT FN.FORM.RESET$("Reset")
```

displays as:

When selected it resets all of the form's input fields to their initial values.

FN.FORM.SELECT\$ Begin or end a FORM input field of TYPE="SELECT". If *name\$* is not a null string, a new SELECT field is started. When *size%* is a nonzero value, it is included in the SELECT field definition to specify the number of option values visible in the list at one time.


When *name\$* is a null string, the SELECT field is terminated.

FN.FORM.SELECT.OPTION\$ Used between a begin SELECT input field and an ending SELECT input field, defines one of the options for the SELECT field.

FN.FORM.SUBMIT\$ Defines a FORM input field of TYPE="SUBMIT". Similar to the [FN.FORM.BUTTON\\$](#) function, uses the value of *label\$* for the label of the button. However, no name/value pair is included when the button is used to submit the form. Use the [FN.FORM.BUTTON\\$](#) function if this information is needed.


This form of the submit button is used when a single submit buttons is displayed in a form.

```
PRINT FN.FORM.SUBMIT$ ("Submit")
```

displays as: 

FN.FORM.TEXTAREA\$ Defines a FORM input field of TYPE="TEXTAREA".

```
PRINT FN.FORM.TEXTAREA$ ("item",3,20,"some text")
```

displays as: 

When selected it defines `item="some text"`.

Returns: These functions generate and return a string containing the generated HTML text line.

Errors: No errors are detected or reported by these functions.

Notes: These CGI functions generate HTML code for standard, simple forms. Many of the FORM tags support attributes in addition to the standard ones coded by these functions. For instance, INPUT fields have an ALIGN= attribute that is not generated by these functions. Also, there are additional INPUT fields supported by HTML with no corresponding CGI function to create them. For instance, input types PASSWORD, IMAGE and FILE fields.

The full specifications for FORM and INPUT tags can be found on the Internet by accessing

`"http://www.w3.org/TR/REC-html40"` (HTML 4.0 specification)

or

`"http://www.w3.org/TR/REC-html32"` (HTML 3.2 specification).

FN.FORM\$ This function should be used in pairs to begin and end a FORM definition. Between the pairs the other functions are used to define the various input fields for the form.

program\$ is either a null string, a "mailto" address such as "mailto:foo@bar.com," or the URL for the CGI program used to process the response from the client when this FORM is submitted. The CGI program does not have to exist on this server.

When *program\$* is not a null string, the text generated is:

```
<FORM ACTION="program$" METHOD="POST">
```

When *program\$* is a null string, the text generated is:

```
</FORM>
```

FN.FORM.CHECKBOX\$ When *checked%* is not zero, the text generated is:

```
<INPUT NAME="name$" TYPE="CHECKBOX" CHECKED  
VALUE="value$"> text$
```

or, when *checked%* is zero:

```
<INPUT NAME="name$" TYPE="CHECKBOX"  
VALUE="value$">text$
```

FN.FORM.HIDDEN\$ The text generated is:

```
<INPUT NAME="name$" TYPE="HIDDEN" VALUE="value$">
```

FN.FORM.INPUT\$ The text generated is:

```
<INPUT NAME="name$" TYPE="TEXT" SIZE="STR$(size%) "  
VALUE="value$">
```

FN.FORM.PASSWORD\$ The text generated is:

```
<INPUT NAME="name$" TYPE="PASSWORD"  
SIZE="STR$(size%) " VALUE="value$">
```

FN.FORM.RADIO\$ When *checked%* is not zero, the text generated is:

```
<INPUT NAME="name$" TYPE="RADIO" CHECKED  
VALUE="value$"> text$
```

or, when *checked%* is zero:

```
<INPUT NAME="name$" TYPE="RADIO" VALUE="value$">  
text$
```

FN.FORM.RESET\$ The text generated is:

```
<INPUT TYPE="RESET" VALUE="label$">
```

FN.FORM.SELECT\$ This function should be used in pairs to begin and end a SELECT field definition. Between the pairs the **FN.FORM.SELECT.OPTION\$** function is used to define the selection values available.

When *name\$* is not a null string, the text generated is:

```
<SELECT NAME="name$" SIZE="STR$(size%)">
```

or, if *size%* is zero:

```
<SELECT NAME="name$">
```

When *name\$* is a null string, the text generated is:

```
</SELECT>
```

FN.FORM.SELECT.OPTION\$ This function should only be used between a pair of beginning and ending **FN.FORM.SELECT\$** function calls. It defines one of the values for the SELECT field and generates the text line:

```
<OPTION VALUE="value$" SELECTED> text$
```

or, when *selected%* is zero:

```
<OPTION VALUE="value$"> text$
```

Multiple **FN.FORM.SELECT.OPTION\$** functions call are normally used between the beginning and ending **FN.FORM.SELECT\$** function calls with each one defining one of the values that may be selected by the user. Only one of these functions should use a nonzero value for *selected%*.

FN.FORM.SUBMIT\$ The text generated is:

```
<INPUT TYPE="SUBMIT" VALUE="label$">
```

FN.FORM.TEXTAREA\$ The text generated is:

```
<TEXTAREA NAME="name$" ROWS="STR$(rows%)"
  COLS="STR$(size%)"> text$ </TEXTAREA>
```

The value of *text\$* is used by HTTP client to initialize the contents of the input field.

Restrictions: The [CGI.INIT](#) function must be used in your program because it defines sub-functions used by these functions.

FN.FORM.CHECKBOX\$, FN.FORM.HIDDEN\$, FN.FORM.INPUT\$, FN.FORM.RADIO\$ Although not tested by this function, these fields must have a *name\$* value and a *value\$* value.

FN.FORM.RADIO\$ Only one RADIO input field in a group of RADIO fields with the same *name\$* value should have *checked%* enabled.

FN.FORM.SELECT.OPTION\$ Although not tested by this function, it is an HTML error for more than one OPTION value to be selected for a SELECT field.

See also: [FN.CGI.GET\\$](#), [FN.CGI.GET.FORM\\$](#)

FN.HTML functions

This set of functions generates common and frequently-used HTML code.

FN.HTML.BREAK\$

FN.HTML.CENTER\$ (*text*\$)

FN.HTML.DECODE\$ (*text*\$)

FN.HTML.ENCODE\$ (*text*\$)

FN.HTML.HEADINGS\$ (*number*%, *text*\$)

FN.HTML.IMAGE\$ (*url*%, *iwidth*%, *iheight*%, *border*%, *alt*\$)

FN.HTML.LINK\$ (*text*\$, *url*\$)

FN.HTML.LIST.ITEM\$

FN.HTML.MAILTO\$ (*text*\$, *url*\$)

FN.HTML.PARA\$

FN.HTML.RULE\$ (*width*%)

FN.HTML.TAG\$ (*tag*\$, *OnOff*%)

<i>alt</i> \$	»	Alternate text for non-image display
<i>border</i> %	»	Size of image border, in pixels
<i>iheight</i> %	»	Height of image, in pixels
<i>iwidth</i> %	»	Width of image, in pixels
<i>email</i> \$	»	E-mail address
<i>number</i> %	»	Heading level number
<i>OnOff</i> %	»	Boolean indicator for begin or end tag.
<i>tag</i> \$	»	Text identifying HTML tag
<i>text</i> \$	»	Text string to display
<i>url</i> \$	»	<u>U</u> niform <u>R</u> esource <u>L</u> ocator for item
<i>width</i> %	»	Percentage width of horizontal rule.

Operation: **FN.HTML.BREAK\$** Generates the HTML code for a paragraph break.

FN.HTML.CENTER\$ Generates the HTML code needed to center *text\$*.

FN.HTML.DECODE\$ The string *text\$* is examined and interpreted as a string coded with Latin-1 characters or character names and decoded into the character set used by THEOS.

A string generated by **FN.HTML.ENCODE\$** function is translated back to the original string that was given to the **FN.HTML.ENCODE\$** function.

FN.HTML.ENCODE\$ Each character in the string *text\$* is examined. Each character matching one of the characters supported by HTML character codes (see “[HTML Character Code Encoding](#)” on page 183) is translated to the corresponding code. If the character is a control character other than a tab, carriage return or new line, it is deleted.

FN.HTML.HEADNG\$ Generates the HTML code for a level-*number%* heading.

FN.HTML.IMAGE\$ Generates the HTML code for an in-line image of *url\$*.

The *iwidth%* and *iheight%* values may be specified as zero, indicating that the image’s natural width or height is to be used when it is displayed.

FN.HTML.LINK\$ Generates the HTML code to mark *text\$* as a hyperlink reference to *url\$*.

FN.HTML.LIST.ITEM\$ Generates the HTML code for a list item.

FN.HTML.MAILTO\$ Generates the HTML code for a hyperlink reference of *text\$* to a “mailto” reference of *email\$*.

FN.HTML.PARA\$ Generates the HTML code for a new paragraph.

FN.HTML.RULE\$ Generates the HTML code for a centered, horizontal rule of *width%*. The value of *width%* is interpreted as a percentage of the width of the user agents window or the enclosing table cell if used in a table.

FN.HTML.TAG\$ Generates the HTML starting or ending character tag for *tag\$*. When *OnOff%* is not zero, a starting tag is generated; when it is zero, an ending tag is created.

The specific tag generated is defined by *tag\$*. Only valid tags are allowed and the generated tag is the standard abbreviation for the *tag\$* specified. For instance, specifying a *tag\$* of “BOLD” generates a tag for “B.” Refer to the notes section for a listing of the tags supported by this function.

Returns: These functions generate and return a string containing the generated HTML text line.

Errors: No errors are detected or reported by these functions.

Notes: Many of these function generate trivial code but are provided because the code generated is common to many web pages and the function provides a consistency in the API.

FN.HTML.BREAK\$ The text string generated is:

`
`

The paragraph break element is similar to a “soft carriage-return.” It terminates the current line of text and starts a new line without terminating the current paragraph. Two or more `
` tags in a row cause one or more blank lines to be output.

FN.HTML.CENTER\$ The text string generated is:

`<CENTER>text$</CENTER>`

FN.HTML.DECODE\$ See Table 6: “[HTML Character Code Encoding](#)” on page 183 for a list of the characters translated by this function.

Use the function on all text received from the client if the text will be displayed on this system now or saved in a file that might be displayed at a later time.

You do not have to decode text that was retrieved with the [FN.CGI.GET\\$](#), [FN.CGI.GET.COOKIE\\$](#) or [FN.CGI.GET.FORM\\$](#) functions. Those functions decode the text before returning it.

FN.HTML.ENCOD\$ See Table 6: “[HTML Character Code Encoding](#)” on page 183 for a list of the characters translated by this function.

This function should be used on any string that might contain one of the special characters. The HTML character set does not match the THEOS character set codes nor, for that matter, any other operating system’s standard character code set.

FN.HTML.Heading\$ The text string generated is:

```
<Hn>text$</Hn>
```

Where *n* is STR\$(*number%*)

For instance, FN.HTML.Heading\$(3,"Sub heading level 3") generates:

```
<H3>Sub heading level 3</H3>
```

The heading levels should be used to indicate the hierarchy of the heading. Generally, a browser will display each heading level in a smaller font than the one above it. For instance, an <H1> heading will be larger than an <H2> heading which is larger than an <H3> heading, *etc.*.

FN.HTML.Image\$ The text string generated is:

```
<IMG SRC="url$" WIDTH="STR$(iwidth%)"
HEIGHT="STR$(iheight%)" BORDER="STR$(border%)"
ALT="alt$">
```

If *iwidth%* or *iheight%* are zero, their terms are omitted from the text string. For instance, FN.HTML.Image\$("URL", 0, 0, 0, " ") generates:

```
<IMG SRC="url$" BORDER="0" ALT=" ">
```

Images are frequently used as a hyperlink to another web page. This can be coded with:

```
FN.HTML.Link$("some.site",FN.HTML.Image$
("link.gif",0,0,0,"Link to some.site"))
```

which generates:

```
<A HREF="some.site"><IMG SRC="link.gif" BORDER="0"
ALT="Link to some.site"></A>
```

The *alt\$* text is displayed by text-only browsers and by graphical browsers who have image downloading turned off. A meaningful *alt\$* text should be used for all images. Use a null string or a string containing a single space character only when the image has no meaningful value. For instance, an image used as a graphical bullet.

FN.HTML.LINK\$ The text string generated is:

```
<A HREF="url$">text$</A>
```

This creates a hyperlink in the document to *url\$*. When *text\$* is selected, the *url\$* is retrieved.

url\$ may be a URL specifying the path and file name of a document or a URL specifying the path and file name of a document and a named anchor in that document. When no named anchor is specified, the document is retrieved and the user is positioned to the top of the document.

When a named anchor is specified in *url\$*, the document is retrieved and the user is positioned to the named anchor location. Named anchors are defined in a document with an anchor tag:

```
<A name="anchorname">some text</A>
```

To link to this location you would create a hyperlink with a *url\$* ending with the hash-mark character (#) and the anchor name:

```
PRINT FN.HTML.LINK$("Go to link", "#anchorname")
```

Since there is no path or document name in the *url\$* it is a reference to the current document.

FN.HTML.LIST.ITEM\$ The text string generated is:

```
<LI>
```

A list item is valid within an enclosing or block. (Although valid within a <DIR> or <MENU> blocks, these items are not part of the pending new standard for HTML.)

FN.HTML.MAILTO\$ The text string generated is:

```
<A HREF="mailto:email$">text$</A>
```

FN.HTML.PARA\$ The text string generated is:

```
<P>
```

The paragraph element is optional in an HTML document. Any blank line will terminate the preceding paragraph and start a

new paragraph. However, HTML is a markup language and you should identify or markup the elements of the document, including each paragraph.

FN.HTML.RULE\$ The text string generated is:

```
<CENTER><HR WIDTH="STR$(width%)"></CENTER>
```

or, if *width%* is less than or equal to zero or greater than or equal to 100:

```
<HR>
```

FN.HTML.TAG\$ The text string generated is:

```
<tag$>
```

or

```
</tag$>
```

Only valid character tags are allowed. A null string is returned if the value of *tag\$* is not one of the following:

B	CODE	OL	SMALL	UL
BIG	EM	P	STRONG	UNDERLINE
BOLD	I	PARA	TABLE	VAR
CENTER	ITALIC	PRE	TT	
CITE	KBD	SAMP	U	

These tags all require closing tags and are frequently used on short text phrases. You might define your own function that uses this function to enclose a text phrase within the proper markup. For instance:

```
DEF FN.HTML.PHRASE$(TAG$, TEXT$)
  LOCAL WORK$
  WORK$ = FN.HTML.TAG$(TAG$, 1)
  WORK$ = WORK$&FN.HTML.ENCODE$(TEXT$)
  FN.HTML.PHRASE$ = WORK$&FN.HTML.TAG$(TAG$, 0)
FNEND
```

Using this function generates a string containing both the starting and closing tags around the encoded text.

```
PRINT FN.HTML.PHRASE$("STRONG", "Important:");
      FN.HTML.ENCODE$(TEXT$)
```

outputs:

```
<STRONG>Important:</STRONG>contents of encoded TEXT$
string
```

Restrictions: The `CGI.INIT` function must be used in your program because it defines sub-functions used by these functions.

FN.HTML.HEADINGS The value of *number%* must be in the range 1–6.

Value	Display	HTML code	Value	Display	HTML code
01	©	©	211	Ů	Ü
02	®	®	212	ü	ü
34	"	"	213	û	û
38	&	&	214	ù	ù
60	<	<	215	ú	ú
62	>	>	216	Ç	Ç
192	Ä	Ä	217	ç	ç
193	ä	ä	218	Ñ	Ñ
194	â	â	219	ñ	ñ
195	à	à	220	Æ	Æ
196	á	á	221	æ	æ
197	É	É	222	Å	Å
198	ë	ë	223	å	å
199	ê	ê	224	ß	ß
200	è	è	225	ı	¿
201	é	é	226	ı	¡
202	ï	ï	227	¢	¢
203	î	î	228	£	£
204	ì	ì	229	¥	¥
205	í	í	231	¤	¤
206	Ö	Ö	232	¼	¼
207	ö	ö	233	½	½
208	ô	ô	234	ÿ	ÿ
209	ò	ò	235	§	§
210	ó	ó	237	²	²

Table 6: HTML Character Code Encoding

FN.JAVA functions

This set of functions can be used in addition to or instead of the FN.FORM functions described earilier. They allow you to use the various attributes of the tags that access Java-Script functions that may be defined in your document.

CGI

FN.JAVA.FORM\$ (*program\$ name\$, submit\$ reset\$*)

FN.JAVA.BUTTON\$ (*name\$, label\$, focus\$, blur\$, change\$, mouseover\$, mouseout\$*)

FN.JAVA.CHECKBOX\$ (*name\$, checked%, value\$, text\$, focus\$, blur\$, change\$, mouseover\$, mouseout\$*)

FN.JAVA.INPUT\$ (*name\$, size%, maxlength%, value\$, focus\$, blur\$, change\$, mouseover\$, mouseout\$*)

FN.JAVA.PASSWORD\$ (*name\$, size%, maxlength%, value\$, focus\$, blur\$, change\$, mouseover\$, mouseout\$*)

FN.JAVA.RADIO\$ (*name\$, checked%, value\$, text\$, focus\$, blur\$, change\$, mouseover\$, mouseout\$*)

FN.JAVA.RESET\$ (*label\$, focus\$, blur\$, change\$, mouseover\$, mouseout\$*)

FN.JAVA.SELECT\$ (*name\$, size%, focus\$, blur\$, change\$*)

FN.JAVA.SUBMIT\$ (*label\$, focus\$, blur\$, change\$, mouseover\$, mouseout\$*)

FN.JAVA.TEXTAREA\$ (*name\$, size%, rows%, text\$, focus\$, blur\$, change\$, mouseover\$, mouseout\$*)

<i>blur\$</i>	»	script invoked when field loses focus
<i>change\$</i>	»	script invoked when value of field changes
<i>focus\$</i>	»	script invoked when field gains focus
<i>mouseout\$</i>	»	script invoked when mouse pointer leaves field
<i>mouseover\$</i>	»	script invoked when mouse pointer over field

Operation: These functions are identical to the non-JavaScript forms described on page 170. The argument list for these functions is the same as those standard functions with additional arguments added to the end. These additional arguments allow you to specify the scripts that are to be invoked when certain events occur.

FN.TABLE functions

This set of functions generate text lines that define a TABLE and its contents.

FN.TABLE\$ (*begin%*, *border%*)

FN.TABLE.DATA\$ (*text\$*)

FN.TABLE.HEAD\$ (*text\$*)

FN.TABLE.RECORD\$ (*text\$*)

<i>begin%</i>	»	Boolean indicator for beginning or ending the TABLE
<i>border%</i>	»	Width of TABLE border
<i>text\$</i>	»	Contents of a table data cell, heading cell or record

Operation: **FN.TABLE\$** Begin or end a <TABLE> definition. If *begin%* is not zero, a new <TABLE> definition is started with the BORDER= term specified as *border%*.

When *begin%* is zero, the <TABLE> definition is terminated.

FN.TABLE.DATA\$ Defines a Table Data item.

FN.TABLE.HEAD\$ Defines a Table Head item.

FN.TABLE.RECORD\$ Defines a Table Record item. A table record is a container for one or more table data items or table head items.

Returns: These functions generate and return a string containing the HTML text line.

Errors: No errors are detected or reported by these functions.

Notes: These CGI functions generate HTML code for standard, simple tables. Many of the table tags support additional attributes not generated by these functions. For instance, <TABLE> fields may have an ALIGN= attribute that is not generated by these functions.

There are other table-related elements not supported by this CGI API including the <CAPTION> element and the HTML 4.0 <COLGROUP>, <TBODY>, <TFOOT> and <THEAD> elements.

Tables are also used in HTML 3.2 documents for layout purposes. Technically, this is an abuse of the markup language and some older versions of browsers may not be able to display the information because they do not support the <TABLE> element. Nevertheless, tables are used for this purpose because HTML 3.2 does not provide a good alternative.

The HTML 4.0 standard and Cascading Style Sheets, Level 1 and the draft of Level 2 provide better alternatives. However, most browsers do not support these capabilities yet.

The full specifications for <TABLE> tags can be found on the Internet by accessing

“<http://www.w3.org/TR/REC-html40>” (HTML 4.0 specification)

or

“<http://www.w3.org/TR/REC-html32>” (HTML 3.2 specification).

FN.TABLE\$ This function is used in pairs to begin and end a <TABLE> definition. Between the pairs the other functions are used to define heading cells, rows and table cells.

When *begin%* is not zero, the text line generated is:

```
<TABLE BORDER="STR$(border%)">
```

When *begin%* is zero, the text line generated is:

```
</TABLE>
```

FN.TABLE.DATA\$ The text line generated is:

```
<TD>text$</TD>
```

Table data cells are normally defined within a table record.

FN.TABLE.HEAD\$ The text line generated is:

```
<TH>text$</TH>
```

Table heading cells are normally defined within a table record.

FN.TABLE.RECORD\$ The text line generated is:

```
<TR>text$</TR>
```

Because table records normally contain one or more data cells, it is common for *text*\$ to be the result of a call to one or more [FN.TABLE.DATA\\$](#) functions or [FN.TABLE.HEAD\\$](#) functions.

For instance:

```
PRINT FN.TABLE.RECORD$(FN.TABLE.DATA$("cell1")&
FN.TABLE.DATA$("cell2"))
```

Restrictions: The [CGI.INIT](#) function must be used in your program because it defines sub-functions used by these functions.

FN.URL.DECODE\$, FN.URL.ENCODER\$

These two functions convert a plain text string to a URL-encoded string or vice versa. URL-encoded strings are used when passing information from or to an HTML document or CGI script.

```
FN.URL.DECODE$ ( text$ )
```

```
FN.URL.ENCODER$ ( text$ )
```

text\$ » String to decode or encode

Operation: **FN.URL.DECODE\$** Converts a URL-encoded string to a plain-text string.

FN.URL.ENCODER\$ Converts a plain-text string to a URL-encoded string.

Returns: These functions return the converted form of *text*\$.

Errors: No errors are detected or reported by these functions.

Notes: **FN.URL.DECODE\$** The following conversions are performed:

- Plus-sign converted to space character
- “%hh” converted to equivalent byte character
- CR,LF converted to new-line character

Other characters are left unchanged.

FN.URL.ENCODER\$ The following conversions are performed:

- Space characters converted to plus-sign character
- Letters, digits, \$, -, _ and period characters are unchanged.
- Other characters are converted to “%hh” where *hh* is the hexadecimal value of the character.

Restrictions: The [CGI.INIT](#) function must be used in your program because it defines sub-functions used by these functions.

H: ASP Reference

An Active Server Page is an HTML documents containing ASP statements and expressions that are interpreted by the server prior to delivering the document to a user-agent (browser).

■ General Syntax

An Active Server Page contains a mixture of HTML tags, text and ASP text. An ASP text item is only recognized when it is contained within the opening and closing ASP identifiers (“<%” and “%>”). For instance:

```
<HTML>
<HEAD>
<TITLE>This is the title of the page</TITLE>
</HEAD>
<BODY>
<% For I = 1 to 5 %>
    <A href="page<%=I%>.htm">Jump to page <%=I%></A><BR>
<% Next %>
</BODY>
</HTML>
```

In the above code, all of the text outside of the <% and %> is merely passed to the HTML client (the browser). When the first <% is encountered the server parses the enclosed statement and performs it by repetitively outputting the “<A href ...
” text while incrementing the value in the variable i by one each time. Each time that “<%=i%>” is encountered it is replaced with the current value of the variable i.

The above example is identical to the non-ASP code:

```
<HTML>
<HEAD>
<TITLE>This is the title of the page</TITLE>
</HEAD>
<BODY>
    <A href="page1.htm">Jump to page 1</A><BR>
    <A href="page2.htm">Jump to page 2</A><BR>
    <A href="page3.htm">Jump to page 3</A><BR>
    <A href="page4.htm">Jump to page 4</A><BR>
    <A href="page5.htm">Jump to page 5</A><BR>
</BODY>
</HTML>
```

■ White Space and Carriage Returns

White space (spaces, tabs, carriage returns and line-feeds) can be used freely throughout an ASP statement with a few exceptions.

- ▶ The “<” and the “%” characters and the “%” and the “>” characters must not be separated by any white space characters.
- ▶ White space may not be used within a statement keyword or variable name.

A carriage return must be used following the Then keyword of the multi-line form of the [If \(Mode 2\)](#) described on page 214. A carriage return must not be used following the Then keyword of the single-line form of the [If \(Mode 1\)](#) and it must be used following the statement or text object of that single-line If statement.

For instance, the following two examples are equivalent:

```
<% If version > 4 Then Response.WriteLine("Some text")%>
<% If version > 4 Then %>
    Response.WriteLine("Some text")
<% End If %>
```

Second example:

```
<%if a > 4      then    %>      show      this
<%if
  version <>    "3.0"
  then %>
  Output      this  when
              version  is    not    3.0
              <%      end    if    %>
```

■ Casemode

The casemode of ASP statement keywords, function names and variable names is not significant. For instance: “While,” “while” and “WHILE” are all equivalent and refer to the [While](#) keyword.

■ Comments

To add a comment to an ASP statement use the single quote character. All characters are ignored from the single quote character to the end of the current line or the statement terminator %>, whichever occurs first.

```
<%                ' begin ASP loop
i = 1              ' loop control starts at one
While i < 20        ' repeat following loop
    Response.WriteLine("Loop counter = " & i)
    i = i+2          ' increment loop control by two
loop               ' repeat loop %>
```

■ Expressions

An expression in an ASP statement is much like an expression in any other language. It may be a simple constant, variable name or function reference or it may be a combination of a mixture of constants, variables and functions joined together by operators.

• Constants

An ASP constant may be an integer literal, a string literal, a Boolean literal or a date string literal. Integer and string literals are specified like integers and string literals in other languages: integers with digits only and strings enclosed with a pair of double quotation marks.

Boolean literals include `True` and `False`.

Date literals are specified as formatted date and or times enclosed within a pair of “hash marks.” For instance: `#Jan 23, 2003#`. A date may be specified with abbreviated or fully-spelled-out month name. Dates may be specified with many reasonable formats such as:

```
#Jan 1, 2012#
#12 February 1996#
#02/12/96#
#12Feb1996#
#12:15#
#1:30pm#
#Jan 1, 1999 1:00:15am#
```

Numbers separated by colons are interpreted as time specifications. The first number (other than time values) whose value is between 1 and 31 is interpreted as a day number, larger numbers are year numbers. Year numbers less than 100 have the current century added to them.

- **Variable Names**

Variable names are tokens starting with a letter and containing only uppercase and lowercase letters, digits, periods and underscore characters. Variable names are not case sensitive. A variable name may not be an ASP keyword such as a statement name or function name.

Variables may contain several types of data including: integers, strings, dates and Boolean values. When used in an expression, the type of data in a variable is automatically converted to the proper type for the expression or assignment being performed.

- **Integer**

Integers are maintained with 32-bit integer data items. This means that they may have values in the range of $\pm 2,147,483,648$

- **Boolean**

A Boolean value for a variable or expression may have one of two values: True or not True and are expressed as True or False. When converted to or from an integer or date, any zero value is interpreted as False and any nonzero value is True. When converted to a string a Boolean value is expressed as either "True" or "False."

The value of a Boolean converted from a string value depends upon the contents of the string. Strings containing only numeric digits are evaluated and, if the value is zero the Boolean equivalent of False is used, otherwise the Boolean value of True is used. A string containing characters other than digits always evaluates to a True value while a null or empty string always evaluates to a False Value.

- **Date and Time**

Date values are maintained in a 32-bit integer as the number of seconds since midnight of January 1, 1970. This means that they may have values in the range of 0 — 4,294,967,296 seconds. When expressed as a date, this range corresponds to midnight of January 1, 1970 (base date) through February 5, 2106.

Date values less than 172,800 (48 hours) are interpreted as time-of-day without a date. (The `FormatDateTime` function described on page 209 can express these small values as dates.)

- **String**

String values for variables or expressions may contain very long strings. However, except for very unusual requirements, strings should be kept smaller than 511 characters because that is the limit for line lengths in an HTML document.

- **Operators**

The following operators may be used in expressions:

Operator	Meaning/Function
-	Unary negative
*	Numeric multiply
/	Numeric division
Mod	Numeric modulo
+	Numeric addition
-	Numeric subtraction
&	String concatenation
=	Equal to
<>	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
Not	Logical NOT
And	Logical AND
Or	Logical OR
Xor	Logical XOR
Eqv	Logical equivalence
Imp	Logical implies

Table 7: ASP Expression Operators

■ Statements

variable = *expression*
 = *expression*
 Do...Loop
 Do...Loop Until
 Do...Loop While
 Do Until...Loop
 Do While...Loop
 Exit Do
 Exit For
 For...Next
 If
 If...End If
 Select Case Case...End Select
 Set *variable* = *object*
 While...Wend

■ Functions

Abs (*expression*)
 CBool (*expression*)
 CDate (*expression*)
 Chr (*expression*)
 CInt (*expression*)
 CStr (*expression*)
 Date
 Day (*DateTime-expression*)
 FormatDateTime (*date-variable*, *format-string*)
 Hour (*DateTime-expression*)
 InStr (*string-expression*₁, *string-expression*₂)
 Lcase (*string-expression*)
 Left (*string-expression*)
 Len (*string-expression*)
 Ltrim (*string-expression*)
 Mid (*string-expression*, *start*, *length*)
 Minute (*DateTime-expression*)
 Month (*DateTime-expression*)
 Now
 Right (*string-expression*)
 Rtrim (*string-expression*)
 Second (*DateTime-expression*)
 String (*count*, *string-expression*)
 Time
 Trim (*string-expression*)
 Ucase (*string-expression*)
 WeekDay (*DateTime-expression*)
 Year (*DateTime-expression*)

■ **Objects**

[FileSystemObject](#)
[Request](#)
Request.Cookies
Request.Form
Request.QueryString
Request.ServerVariables
[Response](#).Cookies
Response.Redirect
Response.Write
Response.WriteLine
[Set](#)
[TextStream](#)

■ **Methods**

[Close](#)
CreateObject
[CreateTextFile](#)
[OpenTextFile](#)
[Read](#)
ReadAll
ReadLine
[Server](#).HTMLEncode
Server.MapPath
Server.URLEncode
[Skip](#)
SkipLine
[Write](#)
WriteBlankLines
WriteLine

■ **Properties**

[AtEndOfLine](#)
[AtEndOfStream](#)
Column
Line

Assignment and Reference Statements

The Request function is used to get the value of a variable set by the user-agent or the server.

1 `<% variable = expression %>`

2 `<% = expression %>`

expression »

variable » Name of variable receiving value of expression

Operation: **Mode 1**—This is the basic variable assignment statement.

```
<% NextYear = Year + 1 %>
<% Limit = 25 %>
<% Default = Date %>
<% CurTime = Now %>
```

Mode 2—A statement that starts with the equal sign causes *expression* to be evaluated and inserted into the HTML document.

```
<HTML><BODY>
The current time is <% = Now %>.
</BODY></HTML>
```

The above ASP document might display as:

The current time is July 31, 1998 11:41 AM.

Restriction: Do not use this statement to assign an object reference to a variable. The [Set](#) statement must be used in that situation.

See also: [Set](#)

AtEndOfLine Property

AtEndOfStream Property

These two properties of a [TextStream](#) object refer to the input/output position for an open text file.

1 *ts-object*.**AtEndOfLine**

2 *ts-object*.**AtEndOfStream**

ts-object » A [TextStream](#) object variable name

Operation: **Mode 1**—The **AtEndOfLine** property is true when the input pointer for the text file is immediately after the end of a text record; otherwise it is false.

Mode 2—The **AtEndOfStream** property is true when the end of the text file has been read or when *ts-object* is not the variable name of an open file.

Notes: These properties are initialized to False when a file is first opened.

Restriction: *ts-object* must be the name of an open [TextStream](#) object.

See also: [Column](#), [Line](#), [Read](#), [ReadAll](#), [ReadLine](#), [Skip](#), [SkipLines](#)

Example:

```
Set fs = Server.CreateObject("Scripting.FileSystemObject")
Set file1 = fs.OpenTextFile("my.file:d")
Do While file1.AtEndOfLine <> True
    char = file1.Read(1)
    ...
Loop
file1.Close

Set fs = Server.CreateObject("Scripting.FileSystemObject")
Set file2 = fs.OpenTextFile("my.file2")
Do While Not file2.AtEndOfStream
    text = file2.ReadLine
    ...
Loop
file2.Close
```

ASP

Close Method

This method closes an open [TextStream](#) object.

1 *ts-object*.**Close**

ts-object » A [TextStream](#) object variable name

Operation: The TextStream file specified by *ts-object* is closed.

Notes: No error occurs if *ts-object* is not open.

See also: [CreateTextFile](#), [OpenTextFile](#)

Example: Set fs = Server.CreateObject("Scripting.FileSystemObject")
 Set file1 = fs.OpenTextFile("my.file:d",ForReading)
 ...
 file1.**Close**

Column Property

Line Property

These two properties of a [TextStream](#) object return the character position in a record or the record number for the next input operation on the file.

1 *ts-object*.**Column**

2 *ts-object*.**Line**

ts-object » A [TextStream](#) object variable name

Operation: These properties of a [TextStream](#) object refer to the column or character position, or to the line number of the next character to be written to or read from the open text stream file.

Notes: When a file is first opened ForReading the Column property is set to zero and the Line property is set to one.

Restriction: *ts-object* must be a variable name assigned to an open [TextStream](#) object. (See [CreateTextFile](#) object or [OpenTextFile](#) object.)

See also: [AtEndOfLine](#), [AtEndOfStream](#), [Read](#), [ReadLine](#), [Skip](#), [SkipLines](#)

Example:

```
Set fs = Server.CreateObject("Scripting.FileSystemObject")
Set file1 = fs.OpenTextFile("my.file:d",ForReading)
fs.SkipLine(5)                    ' Skip first 5 lines
While fs.Line < 20                ' Read lines 6-19
    textline = fs.ReadLine
    ...
Wend
file1.Close
```

CreateTextFile Method

The CreateTextFile method creates a new [TextStream](#) object.

1 **Set** *ts-object* = *fs-object*.CreateTextFile(*file-specification*)

2 **Set** *ts-object* = *fs-object*.CreateTextFile(*file-specification*, *overwrite*)

file-specification » Expression specifying name of file in server system's file system

fs-object » A [FileSystemObject](#) object variable name

overwrite » Optional Bool expression specifying if file replaces existing file

ts-object » A [TextStream](#) object variable name

Operation: The syntax is shown in a [Set](#) statement because that is the only place that a CreateTextFile method is used. Both modes create a [TextStream](#) object.

Mode 1—This method opens an existing stream file or, if *file-specification* is not found, creates a new stream file and opens it.

Mode 2—When a true *overwrite* expression is specified and *file-specification* exists, the existing file is erased and a new file is created and opened. When a false *overwrite* expression is specified the *file-specification* is created and opened only if it does not already exist.

Notes: The *file-specification* is a file name with optional account and path specified. For instance:

```
Set file1 = fs.CreateTextFile("some.file")
Set file1 = fs.CreateTextFile("private\some.file", True)
Set file1 = fs.CreateTextFile("datafile/some.file")
```

This *file-specification* refers to a file in the server system's real file system, not the server's virtual file system. To use a *file-specification* that refers to a file in the virtual file system use the [Server.MapPath](#) object first or as the argument to CreateTextFile.

```
ControlFileName = Server.MapPath("controls/next.item")
Set ControlFile = fs.CreateTextFile(ControlFileName)
```

Restriction: The CreateTextFile method may only be used with a [FileSystemObject](#) object.

See also: [FileSystemObject](#), [OpenTextFile](#)

Example:

```
' filename refers to file in same directory as this ASP document
filename = Server.MapPath("example.textfile")
Set fs = Server.CreateObject("Scripting.FileSystemObject")
Set file1 = fs.CreateTextFile(filename)

For i = 1 To 10
    file1.WriteLine("This is record "&i)
Next

file1.Close

...
```

Do Statement

The Do statement repeats one or more statements until or while a specified condition is true.

- 1 **Do**
statement or text...
Loop
- 2 **Do**
statement or text...
Loop Until *expression*
- 3 **Do**
statement or text...
Loop While *expression*
- 4 **Do Until** *expression*
statement or text...
Loop
- 5 **Do While** *expression*
statement or text...
Loop
- 6 **Exit Do**

<i>expression</i>	»	
<i>statement</i>	»	Another ASP statement
<i>text</i>	»	HTML tags or text included in output

ASP

Operation: **Mode 1**—The simplest form of the Do statement causes the enclosed statements or text to be performed over and over. Termination is not a part of this form of the statement. You must use another statement inside the Do loop to test for some condition and then terminate the Do loop by using the Exit Do statement ([Mode 6](#)).

Mode 2—This form and the Do Until...Loop ([Mode 4](#)) form are similar except that this form performs the enclosed statements one or more times and the [Mode 4](#) form performs the enclosed statements zero or more times.

In this [Mode 2](#) form, the enclosed statements are performed and then *expression* is evaluated and tested. When *expression* is false or zero, the loop is repeated and *expression* is reevaluated.

Mode 3—This form and the Do While...Loop ([Mode 5](#)) form are similar except that this form performs the enclosed statements one or more times and the [Mode 5](#) form performs the enclosed statements zero or more times.

In this [Mode 3](#) form, *expression* is evaluated and tested. When *expression* is true or nonzero, the enclosed statements are performed and *expression* is reevaluated.

Mode 4—This form and the Do...Loop Until ([Mode 2](#)) form are similar except that this form will perform the enclosed statements zero or more times and the [Mode 2](#) form performs the enclosed statements one or more times.

In this [Mode 4](#) form, *expression* is evaluated and tested. When *expression* is false or zero, the enclosed statements are performed and *expression* is reevaluated.

Mode 5—This form and the Do...Loop While ([Mode 3](#)) form are similar except that this form performs the enclosed statements zero or more times and the [Mode 3](#) form performs the enclosed statements one or more times.

In this form, *expression* is evaluated and tested. When *expression* is true or nonzero the enclosed statements are performed and *expression* is reevaluated.

Mode 6—Terminates the Do loop that encloses this statement—control or processing is transferred to the statement or line following the Loop statement. A Do loop may have more than one Exit Do statement.

This statement is ignored if it is not inside of a Do loop statement.

See also: [For](#), [While](#)

Example:

```
Do
    TextRecord = file1.ReadLine
    If file1.AtEndOfStream Then
        Exit Do
    End If
    ... ' process record
Loop
...
Do While Not file1.AtEndOfStream
    TextLine = file1.ReadLine
    ...
Loop
```

FileSystemObject Object

The FileSystemObject object provides access to the server's file system.

Set *fs-object* = **Server.CreateObject**("Scripting.FileSystemObject")

fs-object » A FileSystemObject object variable name

Operation: The syntax is shown in a [Set](#) statement because that is the only place that a FileSystemObject is created.

A FileSystemObject is created and assigned to the *fs-object* variable name.

Notes: FileSystemObject objects are only used with the [CreateTextFile](#) and [OpenTextFile](#) methods.

Restriction: A FileSystemObject must be created with this statement prior to creating or opening any files.

See also: [Server](#)

Example:

```
' filename refers to file in same directory as this ASP document
filename = Server.MapPath("example.textfile")
Set fs = Server.CreateObject("Scripting.FileSystemObject")
Set file1 = fs.OpenTextFile(filename, ForReading)

Do
    text.record = file1.ReadLine
    Response.WriteLine(text.record & "<br>")
    Loop While Not file1.AtEndOfStream

file1.Close

...
```

For Statement

The For statement repeats one or more statements while incrementing and testing a variable each time.

- 1 **For** *variable* = *start-expression* **To** *end-expression*
statement or text...
Next
- 2 **For** *variable* = *start-expression* **To** *end-expression* **Step** *step-expression*
statement or text...
Next
- 3 **Exit For**

<i>end-expression</i>	»	Limiting value of <i>variable</i>
<i>start-expression</i>	»	Initial value of <i>variable</i>
<i>statement</i>	»	Another ASP statement
<i>step-expression</i>	»	Increment (or decrement) value for loop
<i>text</i>	»	HTML tags or text included in output
<i>variable</i>	»	Name of variable used to control loop

Operation:

Mode 1—The *variable* is initialized to *start-expression* value and compared to the value of *end-expression*. If the value of *variable* is less than or equal to the value of *end-expression* then the enclosed *statements or text* are performed. When the **Next** keyword is encountered, *variable* is incremented by one and compared to *end-expression*, etc.

Mode 2—Similar to [Mode 1](#) except that at the end of each iteration of the loop the value of *variable* is modified by adding the value of *step-expression*. Also, if the value of *step-expression* is negative then the comparison is performed by testing to see if the new value of *variable* is greater than or equal to *end-expression*.

Mode 3—This statement is used in a For-Next loop to exit the loop early. Usually, this statement the object of an [If](#). A For-Next loop may have multiple **Exit For** statements. Control is transferred to the statement or text following the **Next** keyword. An **Exit For** is ignored if it is not inside of a For-Next loop.

Notes: For-Next loop statements may be nested if the control *variable* for each loop is different. For instance:

```
<% For i = 1 To 10
    For j = 1 To 10
        For k = 1 To 5
            ...
        Next
    Next
Next %>
```

Restriction: Single-line If statements cannot be used inside of a For-Next statement.

See also: [Do](#), [If](#), [While](#)

Functions

The following function are used in expressions to get special server values or to manipulate other expressions.

- 1 **Abs** (*numeric-expression*)
- 2 **CBool** (*expression*)
- 3 **CDate** (*expression*)
- 4 **Chr** (*expression*)
- 5 **CInt** (*expression*)
- 6 **CStr** (*expression*)
- 7 **Date**
- 8 **Day** (*datetime-expression*)
- 9 **FormatDateTime** (*datetime-expression*, *format-string*)
- 10 **Hour** (*datetime-expression*)
- 11 **InStr** (*start-column*, *string-expression*₁, *string-expression*₂)
- 12 **Lcase** (*string-expression*)
- 13 **Left** (*string-expression*, *length*)
- 14 **Len** (*string-expression*)
- 15 **Ltrim** (*string-expression*)
- 16 **Mid** (*string-expression*, *start*, *length*)
- 17 **Minute** (*datetime-expression*)
- 18 **Month** (*datetime-expression*)
- 19 **Now**
- 20 **Right** (*string-expression*)
- 21 **Rtrim** (*string-expression*)
- 22 **Second** (*datetime-expression*)
- 23 **String** (*count*, *string-expression*)
- 24 **Time**
- 25 **Trim** (*string-expression*)
- 26 **Ucase** (*string-expression*)
- 27 **WeekDay** (*datetime-expression*)
- 28 **Year** (*datetime-expression*)

<i>count</i>	»	Number of times to repeat character
<i>datetime-expression</i>	»	Expression containing date/time value
<i>expression</i>	»	Expression containing Boolean, string, date or numeric value
<i>format-string</i>	»	String expression containing date/time formatting codes
<i>length</i>	»	Number of characters to extract
<i>numeric-expression</i>	»	Expression containing numeric value
<i>start</i>	»	Position of first character to extract
<i>string-expression</i>	»	Expression containing alphanumeric characters

Operation: **Abs** Returns the absolute value of *numeric-expression*.

CBool *expression* is evaluated and returned as a Bool value.

<i>expression</i> type	Value	Result
Integer	0	False
	nonzero	True
String	numeric "0"	False
	numeric nonzero	True
	alphanumeric	True
	empty or null	False
Date	0	True
	nonzero	True

CDate *expression* is evaluated and converted to a date value.

<i>expression</i> type	Value	Result
Integer	< 172,800	Time of day
	> 172,800	Date and time
String	looks like date and/or time (see below)	Number of seconds since 01/01/1970
Bool	False	00:00:00 (UTC)
	True	00:00:01 (UTC)

Dates are maintained as a numeric value but always displayed as a string representation of the date and/or time. Date values smaller than 172,800 (48 hours) represent a time-of-day. Dates larger than this value represent the number of seconds since midnight of January 1, 1970.

The format of a string that is converted to a date value is free format...most reasonable representations of dates and times may be used.

Chr Returns a single character value. If *expression* is numeric, the value of the expression is used, modulo 256, and the corresponding character is returned. If the *expression* is a string, the first character of that string is returned.

CInt *expression* is evaluated and converted to an integer.

<i>expression</i> type	Value	Result
Bool	True	1
	False	0
String	numeric	value of number
	alphanumeric	0
Date	all	Number of seconds since January 1, 1970

CStr *expression* is evaluated and converted to a string.

<i>expression</i> type	Value	Result
Integer	all	string representation of value
Bool	True	"True"
	False	"False"
Date	all	Number of seconds since January 1, 1970

Date Returns the current system date.

Day Returns the day number of the month specified in the *datetime-expression*. Use the **Date** or **Now** functions for the *datetime-expression* for the current day number.

FormatDateTime Returns the *datetime-expression* formatted as a string according to the specifications in *format-string*. *Format-string* contains literal characters and special format characters identical to the format string used by MultiUser BASIC's STRTIME\$ function or C's strftime function. See "Notes" section for *format-string* formatting characters.

Hour Returns the hour number of the day specified in the *datetime-expression*. Use the **Time** or **Now** functions for the *datetime-expression* for the current hour number.

InStr Searches *string-expression*₁ starting at *start-column* and Returns the starting character position of *string-expression*₂ within *string-expression*₁. Both strings are case sensitive. *start-column* may be omitted, in which case the search starts at column one of *string-expression*₁.

Lcase	Returns the lowercase form of <i>string-expression</i> . Only uppercase characters are modified; lowercase and non-alphabetic characters remain as-is in the string returned.
Left	Returns the first <i>length</i> number of characters in <i>string-expression</i> .
Len	Returns the length of <i>string-expression</i> .
Ltrim	Returns the <i>string-expression</i> with leading (left) spaces trimmed off of it.
Mid	Returns <i>length</i> number of characters in <i>string-expression</i> , starting with character position <i>start</i> .
Minute	Returns the minute number of the hour specified in the <i>datetime-expression</i> . Use the Time or Now functions for the <i>datetime-expression</i> for the current minute number.
Month	Returns the month number of the year specified in the <i>datetime-expression</i> . Use the Date or Now functions for the <i>datetime-expression</i> for the current month number.
Now	Returns the current system date and time.
Right	Returns the rightmost <i>length</i> characters from <i>string-expression</i> .
Rtrim	Returns the <i>string-expression</i> with trailing (right) spaces trimmed off of it.
Second	Returns the second number of the minute specified in the <i>datetime-expression</i> . Use the Time or Now functions for the <i>datetime-expression</i> for the current second number.
String	Returns a string of <i>count</i> repetitions of the first character in <i>string-expression</i> .
Time	Returns the current system time.
Trim	Returns the <i>string-expression</i> with leading and trailing spaces trimmed off of it. Also, each embedded multiple spaces are reduced to a single space character.
Ucase	Returns the uppercase form of <i>string-expression</i> . Only lowercase characters are modified; uppercase and non-alphabetic characters remain as-is in the string returned.

- WeekDay** Returns the day number of the week for *datetime-expression*. Day numbers are computed with Sunday as day 1.
- Year** Returns the year number specified in the *datetime-expression*. Use the **Date** or **Now** functions for the *datetime-expression* for the current year number. **Year** always returns a four-digit year.

Notes: These functions are almost identical to their MultiUser BASIC equivalents. Specifically:

ASP Function	MultiUser BASIC Equivalent
<code>Abs(<i>exp</i>)</code>	<code>ABS(<i>exp</i>)</code>
<code>Chr(<i>exp</i>)</code>	<code>CHR\$(<i>exp</i>)</code>
<code>Date</code>	<code>DATE\$(0)</code>
<code>Day(<i>exp</i>)</code>	<code>STRTIME\$("%d",DAY(<i>exp</i>),0)</code>
<code>FormatDateTime(<i>exp</i>, <i>fmt</i>)</code>	<code>STRTIME\$(<i>fmt</i>,DAY(<i>date-exp</i>),SECOND(<i>time-exp</i>))</code>
<code>Hour(<i>exp</i>)</code>	<code>STRTIME\$("%H",0,SECOND(<i>exp</i>))</code> or <code>LEFT\$(TIME\$(<i>exp</i>),2)</code>
<code>InStr(<i>start</i>,<i>exp</i>₁,<i>exp</i>₂)</code>	<code>SCH\$(<i>start</i>,<i>exp</i>₁,<i>exp</i>₂)</code>
<code>InStr(<i>exp</i>₁,<i>exp</i>₂)</code>	<code>SCH\$(0,<i>exp</i>₁,<i>exp</i>₂)</code>
<code>Lcase(<i>exp</i>)</code>	<code>LCASE\$(<i>exp</i>)</code>
<code>Left(<i>exp</i>,<i>len</i>)</code>	<code>LEFT\$(<i>exp</i>,<i>len</i>)</code>
<code>Len(<i>exp</i>)</code>	<code>LEN(<i>exp</i>)</code>
<code>Ltrim(<i>exp</i>)</code>	<code>LTRIM\$(<i>exp</i>)</code>
<code>Mid(<i>exp</i>,<i>start</i>,<i>len</i>)</code>	<code>MID\$(<i>exp</i>,<i>start</i>,<i>len</i>)</code>
<code>Minute(<i>exp</i>)</code>	<code>STRTIME\$("%M",0,SECOND(<i>exp</i>))</code> or <code>MID\$(TIME\$(<i>exp</i>),4,2)</code>
<code>Month(<i>exp</i>)</code>	<code>STRTIME\$("%m",DAY(<i>exp</i>),0)</code>
<code>Now</code>	<code>TIME\$(0)</code>
<code>Right(<i>exp</i>,<i>len</i>)</code>	<code>RIGHT\$(<i>exp</i>,LEN(<i>exp</i>)-<i>len</i>+1)</code>
<code>Rtrim(<i>exp</i>)</code>	<code>RTRIM\$(<i>exp</i>)</code>
<code>Second(<i>exp</i>)</code>	<code>STRTIME\$("%S",0,SECOND(<i>exp</i>))</code> or <code>RIGHT\$(TIME\$(<i>exp</i>),7)</code>

ASP Function	MultiUser BASIC Equivalent
String(<i>count,exp</i>)	RPT\$(<i>count</i> ,LEFT\$(<i>exp</i> ,1))
Time	TIME\$(0)
Trim(<i>exp</i>)	TRIM\$(<i>exp</i>)
Year(<i>exp</i>)	STRTIME\$("%Y",DAY(<i>exp</i>),0)

The *format-string* contains literal characters and translation directives.

Char	Time Element Substitution
%a	Abbreviated weekday name (Sun, Mon, <i>etc.</i>). †
%A	Full weekday name (Sunday, Monday, <i>etc.</i>). †
%b	Abbreviated month name (Jan, Feb, <i>etc.</i>). †
%B	Full month name (January, February, <i>etc.</i>). †
%c	Date and time in standard DATEFORM format. For instance, DATEFORM 1 produces: MM/DD/YY HH:MM:SS. ‡
%C	Date and time in current locale's long format, using "%J %B %Y %H:%M" as in "15 April 1999 15:23."
%d	Day number of month (01-31).
%D	Date in format: MM/DD/YY.
%e	Day number of month (1-31) with single digits preceded by a space character.
%h	Abbreviated month name (Jan, Feb, <i>etc.</i>), synonym to "%b".
%H	Hour number, 24-hour clock (00-23).
%I	Hour number, 12-hour clock (01-12).
%j	Julian day number of year (001-366).
%J	Day number of month (1-31) with single digits not preceded by a space or zero character.
%k	Hour number (0-23) with single digits preceded by a space character.
%l	Hour number (1-12) with single digits preceded by a space character.
%m	Month number (01-12).
†	The SYSTEM.TEOS32.MESSAGES file is used to get the actual names.
‡	The date format is determined by the DATEFORM environment variable.

Table 8: FormatDateTime Directives

Char	Time Element Substitution
%M	Minute number (00-59).
%n	Same as a “\n.”
%p	AM or PM. †
%r	Time in standard 12-hour format, using “%I:%M:%S %p”.
%R	Time in standard 24-hour format, using “%H:%M”.
%S	Second number (00-59).
%t	Tab character.
%T	Time in standard 24-hour format: HH:MM:SS.
%U	Week number of year, Sunday is 1st day of week (00-52).
%w	Weekday number (0=Sunday through 6=Saturday).
%W	Week number of year, Monday is 1st day of week (00-52).
%x	Date in standard format: MM/DD/YY. ‡
%X	Time in standard 24-hour format: HH:MM:SS.
%y	Year number in century (00-99).
%Y	Year number (1900-2106).
%Z	Timezone name.
%%	Literal percent character.
†	The SYSTEM.TEOS32.MESSAGE <i>n</i> file is used to get the actual names.
‡	The date format is determined by the DATEFORM environment variable.

Table 8: FormatDateTime Directives

The conversion functions CBool, CDate, CInt and CStr are used internally whenever a variable or expression needs to be converted to evaluate or assign an expression value.

See also: [Request](#), [Response](#)

If Statement

The If statement conditionally executes ASP statements and functions depending upon the current value of an expression.

1 **If** *expression* **Then** *statement*

2 **If** *expression* **Then**
 statement or text1...
 Else
 statement or text2...
 End If

expression » A Boolean expression value

statement » Another ASP statement

text » HTML tags or text included in output

Operation: **Mode 1**—This is the single-line form of the If statement. *Expression* is evaluated and, if it is true (non-zero) then the *statement* is performed. If the value of *expression* is false (zero) then the *statement or text* is skipped.

The syntax of this statement is different from all other statements because the carriage-return or end-of-statement is important. In this format the *statement* must be on the same line as the If statement and the carriage return character or end-of-statement terminates the statement. That is, when *expression* is true then the *statement* following the Then keyword is performed and conditional processing terminates for this statement.

```
<% If Not Request.Form(limit) Then no.limit = True %>
```

Mode 2—This is the multiline form of the If statement. With this format, nothing must follow the Then keyword (except maybe the statement termination characters %>).

expression is evaluated and, if it is true the *statement or text1* following is performed and the *statement or text2* is skipped. *Statement or text1* is terminated by an Else keyword or End If keywords. When *expression* is false *statement or text1* is skipped and *statement or text2* is performed.

The Else clause is optional and when omitted the *statement or text2* must be also omitted.

Notes: Because SSI statements are processed prior to ASP interpretation of the page, any SSI statements used as the object of an If or Else clause are processed whether or not the *expression* is true.

Restriction: Single-line If ([Mode 1](#)) statements may not be used inside of any [Do](#), [For](#) or [While](#) statement.

See also: [Do](#), [For](#), [Select Case](#), [While](#)

Example:

```
<%  
Set bc = Server.CreateObject("MSWC.BrowserType")  
If bc.TextOnly = True Then  
    Response.Redirect("textver.htm")  
End If %>  
...
```

MSWC.BrowserType

The MSWC.BrowserType argument to the Server.CreateObject object creates an object that contains values of the properties of the referring agent (browser).

1 **Set** *bt-object* = **Server.CreateObject**("MSWC.BrowserType")

2 *bt-object.property*

bt-object

»

Variable name of MSWC.BrowserType object

property

»

Name of MSWC.BrowserType property

Operation: The syntax is shown in a [Set](#) statement because that is the only place that a MSWC.BrowserType object is created.

Mode 1—This method creates a MSWC.BrowserType object for the user agent (browser). The referrer description is used as a key to lookup and set the properties of the object. (See notes below.)

Mode 2—The *property* of the MSWC.BrowserType object specified by *bt-object* is returned.

Errors: In a [Mode 2](#) reference, if *property* does not match one of the properties of a MSWC.BrowserType object or if *bt-object* is not defined by a [Mode 1](#) statement, a null string or false value is used.

Notes: The properties of a MSWC.BrowserType include:

property	Description
Browser	Name of browser
Version	Version of browser
Platform	Operating system platform on browser’s computer
Language	Languages supported by browser
Frames	Boolean value specifying frame support in browser
Tables	Boolean value specifying table support in browser
Cookies	Boolean value specifying cookie support in browser

Table 9: Browser Capabilities

property	Description
BackgroundSounds	Boolean value specifying background sound support in browser
VBScript	Boolean value specifying Visual Basic Script support in browser
JavaScript	Boolean value specifying Java Script support in browser
JavaApplets	Boolean value specifying Java Applet support in browser
ActiveXControls	Boolean value specifying ActiveX support in browser
TextOnly	Boolean value specifying text only browser

Table 9: Browser Capabilities

The values for these properties are determined by a search of the database maintained by the server (SYSTEM.TEOS32.BROWSCAP). New browsers or new versions of browsers may not be found in this database and the information returned will be inaccurate in this situation.

See also: [Server](#)

Example:

```
<% Set bc = Server.CreateObject("MSWC.BrowserType")
If (bc.Browser = "IE" Or bc.Browser = "Netscape") And
    bc.version >= "4.0" Then %>
    <LINK REL="stylesheet" TYPE="text/css" HREF="standard.css">
<% End If %>
```

OpenTextFile Methods

The `OpenTextFile` method opens a text file for input or output.

1 **Set** *ts-object* = *fs-object*.**OpenTextFile**(*file-specification*)

2 **Set** *ts-object* = *fs-object*.**OpenTextFile**(*file-specification*, *mode*, *create*)

<i>create</i>	»	Optional Bool expression specifying if file replaces existing file
<i>file-specification</i>	»	Expression specifying name of file in server's file system
<i>fs-object</i>	»	A FileSystemObject object variable name
<i>mode</i>	»	Optional constance specifying ForReading, ForWriting or ForAppending
<i>ts-object</i>	»	A TextStream object variable name

Operation: The syntax is shown in a [Set](#) statement because that is the only place that an `OpenTextFile` method is used.

Mode 1—This method opens an existing stream file `ForReading` or, if *file-specification* is not found, creates a new stream file and opens it `ForWriting`.

Mode 2—When *create* expression is true and *file-specification* exists, the existing file is erased and a new, empty file is created. A false *create* expression operates like a [Mode 1](#) statement. The file is opened for input (`ForReading`), for output (`ForWriting`) or output append (`ForAppending`) depending upon the value of *mode*.

Notes: The *file-specification* is a file name with optional account and path specified. For instance:

```
Set file1 = fs.OpenTextFile("some.file")
Set file2 = fs.OpenTextFile("private\some.file")
Set file3 = fs.OpenTextFile("datafile/some.file")
```

This *file-specification* refers to a file in the server system's real file system, not the server's virtual file system. To "virtualize" the name, use the [Server.MapPath](#) object first or as the argument to `OpenTextFile`.

```
Set ControlFileName = Server.MapPath("controls/next.item")
Set File4 = fs.OpenTextFile(ControlFileName, ForReading)
```

The properties [AtEndOfLine](#), [AtEndOfStream](#), [Column](#) and [Line](#) are initialized for this newly opened file. [AtEndOfLine](#) and [AtEndOfStream](#) are set to False; [Line](#) is set to one; [Column](#) is set to zero.

Restrictions: Only sequential stream files may be opened.

Defaults: When *create* is not specified it defaults to a false value.

ForReading is used when *mode* is not specified.

See also: [CreateTextFile](#), [Server.MapPath](#)

Examples:

```
Set fs = Server.CreateObject("Scripting.FileSystemObject")
ControlFileName = Server.MapPath("/Problem.control")
Set ControlFile = fs.OpenTextFile(ControlFileName, ForReading)

While Not ControlFile.AtEndOfStream
    text = ControlFile.ReadLine
    Response.WriteLine(text&" ")
Wend

ControlFile.Close
```

Read Methods

The Read methods read characters, a line or all of a [TextStream](#) object file.

1 *ts-object*.**Read**(*count*)

2 *ts-object*.**ReadAll**

3 *ts-object*.**ReadLine**

count » Number of character to read from [TextStream](#) object

Operation: **Mode 1**—*count* number of characters are read from the *ts-object* [TextStream](#) object file.

Mode 2—The entire remainder of the *ts-object* [TextStream](#) object file is read.

Mode 3—Reads characters from the *ts-object* [TextStream](#) object file up to and including the next end-of-line character. The end-of-line characters is discarded.

Returns: **Mode 1**—Characters read from the file.

Mode 2—All of the characters read from the file.

Mode 3—The remaining characters in the current line of the file.

Notes: The properties [AtEndOfLine](#), [AtEndOfStream](#), [Column](#) and [Line](#) are updated to reflect the change in the file's input position.

The ReadAll method should only be used on very small files.

Restriction: *ts-object* must be a [TextStream](#) object file opened with [OpenTextFile](#) and mode ForReading or opened with [CreateTextFile](#) method.

See also: [Assignment](#) statement, [AtEndOfLine](#), [AtEndOfStream](#), [Column](#), [Line](#), [OpenTextFile](#) [Read](#), [Skip](#) and [Write](#) methods

Request Object

The Request object retrieves information from the referring HTML page or agent.

- 1 **Request**(*variable-name*)
- 2 **Request.Cookies**(*variable-name*)
- 3 **Request.Form**(*variable-name*)
- 4 **Request.QueryString**(*variable-name*)
- 5 **Request.ServerVariables**(*variable-name*)

variable-name » Expression specifying name of cookie, form, query-string or server variable

Operation: **Mode 1**—Gets the first value for *variable-name* defined by: a cookie, the form, the query string or server variable. A null string is returned if none of these define a value for *variable-name*.

Mode 2—The value of *variable-name* defined in a cookie on the client agent's system is returned. A null string is returned if *variable-name* is not defined in a cookie on the agent's system.

Mode 3—If this page was invoked by posting a form from another page, the value defined in that form for *variable-name* is extracted and returned. A null string is returned if there is no form data or if *variable-name* is not defined in that form.

Mode 4—If this page was invoked with a URL specifying a query string, the value of *variable-name* defined in that query string is returned. Otherwise, a null string is returned. A null string is also returned if the query string does not define a value for *variable-name*.

Mode 5—Gets the value of the server variable *variable-name*, similar to the SSI `#echo var=` statement described on page [147](#).

Returns: The value of the requested variable replaces the entire function reference in the document.

Notes: The *server-variable* names supported by the THEOS HTTP Server include:

Variable-name	Meaning
AUTH_TYPE	This is the protocol-specific authentication method used to validate the user. For the THEOS HTTP Server, “Basic” is always returned.
CONTENT_LENGTH	Length of POST document
CONTENT_TYPE	GET or POST
DOCUMENT	The path and file name of the current web page, relative to the server.
DOCUMENT_URI	The path and file name of the current web page, relative to the server home directory.
DATE_GMT	The date and time on the server adjusted to Greenwich Mean Time (GTM) which is now called UTC time. This date/time is formatted according to the current #config TimeFmt= setting.
DATE_LOCAL	The date and time on the server in its local time. This date/time is formatted according to the current #config TimeFmt= setting.
GATEWAY_INTERFACE	The revision of the CGI specification to which this server complies. Format: CGI/revision
LAST_MODIFIED	The date and time that the current web page file was last changed. This date/time is formatted according to the current #config TimeFmt= setting.
PATH_INFO	blank always CGI
PATH_TRANSLATED	blank always CGI
QUERY_STRING	If this web page was referenced with a query string specified, this variable contains the text following the “?”. It is not decoded in any fashion.
QUERY_STRING_UNESCAPED	The query string with special characters not escaped.

Table 10: Server Variables

Variable-name	Meaning
REMOTE_ADDR	The IP address of the remote client requesting this web page. (When a proxy server is used between the server and the client, this may be the proxy server IP address.)
REMOTE_HOST	The host name of the remote client requesting this web page. If the server does not have this information, it uses the value of REMOTE_ADDR. (When a proxy server is used between the server and the client, this may be the proxy server host name.)
REQUEST_METHOD	Contains the method used to invoke this document. It is either "GET" or "POST."
SCRIPT_NAME	The virtual path to this web page being executed.
SERVER_NAME	The server's hostname, as defined in the SYSTEM.THEOS32.HTTPDCFG file.
SERVER_PORT	The port number used by the server.
SERVER_PROTOCOL	The name and revision of the information protocol used by the server. Format: protocol/revision, i.e., HTTP/1.0.
SERVER_SOFTWARE	The name and version of the HTTP server software. Format: name/version, i.e., THEO+Server/1.0.

Table 10: Server Variables

Restriction: The value of a cookie might be null even though a cookie of that name is defined by the user-agent. Cookies maintained by a browser have domain, path, expiration date and security properties associated with them. If the domain and path of this document are not included in the domain and path of the cookie, no value is returned. Similarly, if the expiration date of the cookie has passed the browser may return a null string.

Support for cookies and restricting cookies to certain domains, paths and dates is controlled by the user agent and options enabled by the user.

See also: [MSWC.BrowserType](#), [Response](#)

Response Object

The Response object writes data to the document being created by this ASP page.

- 1 **Response.Cookies(*cookie-name*) = *value***
- 2 **Response.Cookies(*cookie-name*).*property* = *value***
- 3 **Response.Redirect(*URL*)**
- 4 **Response.Write(*string*)**
- 5 **Response.WriteLine(*string*)**

<i>cookie-name</i>	»	Expression specifying name of cookie variable
<i>property</i>	»	Cookie property being defined
<i>string</i>	»	String expression to be written to document
<i>URL</i>	»	Expression specifying <u>U</u> niform <u>R</u> esource <u>L</u> ocator for the redirected page
<i>value</i>	»	New value of cookie variable

Operation: **Mode 1**—This form defines the primary value for a cookie.

Mode 2—This form defines the value for one of a cookie's properties.

Mode 3—Specifies that the document is to be automatically redirected to another document.

Mode 4—Write the value of *string* expression to the HTML document

Mode 5—Write the value of the *string* expression to the HTML document and appends a carriage return.

Notes: **Mode 1**—Normally this form is used in combination with Mode 2 objects:

```
Response.Cookies("User") = "My Name"
Response.Cookies("User").Expires = Now+2*86400
Response.Cookies("User").Path = "/"
Response.Cookies("User").Domain = ".mydomain.com"
```


Mode 2—The properties of a cookie include:

Property	Usage
Domain	String expression specifying domain name of this web site
Expires	Expiration date for cookie
Path	Path within domain that cookie applies to
Secure	Boolean expression specifying that cookie is secure

Mode 3—If the *URL* contains any characters other than letters, digits, \$, -, _ (underscore) and period it should be encoded with the [Server.URLEncode](#) method.

Mode 4—This form provides an easy method of writing text and the values of expressions to the document without continually switching between ASP statements and non-ASP statements.

Restriction:

Mode 1—This object may only be used prior to outputting any content for the document being created. It is used prior to the “<HTML>” tag. Multiple cookies and their properties may be defined in a document.

Mode 2—Same restrictions as Mode 1.

Mode 3—This object may only be used prior to outputting any content for the document being created. It is used prior to the <HTML> tag of the document. If there is text in the <HEAD> or <BODY> sections of the document, they will only be displayed by user agents that do not support automatic redirection.

See also:

[Write](#)

Select Case Statement

The Select Case statement processes one set of statements out of many sets depending upon the value of an expression.

```
Select Case test-expression  
    Case expression-list  
        statement-or-text...  
    Case Else  
        statement-or-text...  
End Select
```

<i>expression</i>	»	String, Boolean, DateTime or numeric expression
<i>expression-list</i>	»	<i>select-expression</i> [, <i>expression-list</i>]
<i>range</i>	»	<i>expression</i> To <i>expression</i>
<i>select-expression</i>	»	<i>expression</i> <i>range</i>
<i>test-expression</i>	»	<i>expression</i>
<i>statement</i>	»	Another ASP statement
<i>text</i>	»	HTML tags or text included in output

Operation: *test-expression* is evaluated and compared with each of the *expression-list* expressions for each Case clause. When a match occurs the *statement-or-text* following that Case clause is processed. After processing those *statement-or-text* and a Case or Case Else clause is encountered, all further lines are skipped until the End Select is encountered.

If no *expression-list* expression matches *test-expression*, and a Case Else is encountered, the *statement-or-text* of that Case Else is processed.

Notes: Select Case statements may be nested. That is, a Select Case statement may be the object of the Case or Case Else clause of another Select Case statement.

You may have multiple Case clauses for each Select Case statement.

You should have only one Case Else clause. If multiple Case Else clauses are specified, only the first one encountered is used. The Case Else clause should be the last clause of an If statement structure. When the first Case

Else clause is encountered, the *statement-or-text* is processed if all previous Case clauses were not processed because their expressions were false. Any subsequent Case Else clause is always skipped.

Because SSI statements are processed prior to ASP interpretation of the page, any SSI statements used as the object of a Case or Case Else clause are processed and expanded whether or not ASP processes the clause.

See also: [If](#)

Examples:

```
State = Request.Cookie("State")
Select Case State
  Case "WA", "OR", "ID", "MT"
    Response.WriteLine("Northwest")
  Case "CA"
    Response.WriteLine("California")
  Case Else
    Response.WriteLine("Other")
End Select
```

Server Object

The Server object is used to access server resources.

```

1  Set bt-object = Server.CreateObject( "MSWC.BrowserType" )
2  Set fs-object = Server.CreateObject( "Scripting.FileSystemObject" )
3  Server.HTMLEncode( text )
4  Server.MapPath( file-specification )
5  Server.URLEncode( text )

```

<i>bt-object</i>	»	A MSWC.BrowserType object variable name
<i>file-specification</i>	»	Expression specifying file name in server system's file system.
<i>fs-object</i>	»	A FileSystemObject object variable name
<i>text</i>	»	String expression to encode

Operation: **Mode 1**—This statement creates a [MSWC.BrowserType](#) object for the referring agent browser (HTTP_Referred). The referrer description is used as a key to lookup and set the properties of the object. See “[MSWC.BrowserType](#)” on page 216. This object may only be used in a [Set](#) statement.

Mode 2—A [FileSystemObject](#) is created and assigned to the *fs-object* variable name. [FileSystemObject](#) objects are only used with the [CreateTextFile](#) and [OpenTextFile](#) methods. This object may only be used in a [Set](#) statement.

Mode 3—Each character in *string* is examined. Each character matching one of the characters supported by HTML character codes (see “[HTML Character Code Encoding](#)” on page 183) is translated to the corresponding code. If the character is a control character other than a tab, carriage return or new line, it is deleted.

Mode 4—Virtualizes *file-specification*. That is, *file-specification* is converted to be a reference to a file in the server's virtual file system.

Mode 5—Converts the plain-text *string* to a URL-encoded string.

Returns: **Server.HTMLEncode** Encoded text string for *text* expression.

Server.MapPath Virtualized specification for *file-specification*.

Server.URLEncode Encoded text string for *text* expression.

Notes:

MSWC.BrowserType This object provides a capability that is similar to the [FN.CGI.BROWSER\\$](#) function described on page 166.

Scripting.FileSystemObject At least one object of this type is required for any file system operations such as [CreateTextFile](#) or [OpenTextFile](#).

Server.HTMLEncode This function should be used on any string that might contain one of the special characters. The HTML character set does not match the THEOS character set codes nor, for that matter, any other operating system's standard character code set.

Server.MapPath The *file-specification* should not include a drive-code specification. When specified it is ignored.

file-specifications that start with a “/” character are mapped to the HTTP server's virtual root directory. *file-specifications* that do not start with a “/” are mapped relative to the current document's directory.

For instance, if the server's virtual root = “/www:s” and the document URL = “http://www.myserver.com/Support/Logs/Problem.asp”

```
Server.MapPath("data.file")
    returns "/www/Support/Logs/data.file:s"
```

```
Server.MapPath("/data.file")
    returns "/www/data.file:s"
```

Server.URLEncode The following conversions are performed:

- Space characters converted to plus-sign character
- Letters, digits, \$, -, _ and period characters are unchanged.

Other characters are converted to “%hh” where *hh* is the hexadecimal value of the character.

See also: [CreateTextFile](#), [FileSystemObject](#), [MSWC.BrowserType](#), [OpenTextFile](#), [TextStream](#)

Set Statement

The Set statement is used to assign an object reference to a variable.

```
<% Set variable = object %>
```

object » Object and method specification

variable » Name of variable receiving object

Operation: The *variable* is assigned a pointer to the *object* reference.

Notes: The Set statement must be used when assigning a variable to an object. Do not use the [Assignment](#) statement for assigning object references.

See also: [CreateTextFile](#), [FileSystemObject](#), [MSWC.BrowserType](#), [OpenTextFile](#), [Server](#), [TextStream](#)

Examples:

```
Set Browser = Server.CreateObject("MSWC.BrowserType")
Set filesystem = Server.CreateObject("Scripting.FileSystemObject")
ControlFileName = Server.MapPath("/Problem.control")
Set ControlFile = fs.OpenTextFile(ControlFileName, ForReading)
```

Skip Methods

The Skip and SkipLine methods read and discard characters or a line from a [TextStream](#) object file.

1 *ts-object*.Skip(*count*)

2 *ts-object*.SkipLine

count

» Number of characters to skip

ts-object

» Variable name of [TextStream](#) object

- Operation:

Mode 1—*count* number of characters are skipped from the *ts-object* [TextStream](#) object file.

Mode 2—Skips characters from the *ts-object* [TextStream](#) object file up to and including the next end-of-line character.
- Notes:

The properties [AtEndOfLine](#), [AtEndOfStream](#), [Column](#) and [Line](#) are updated to reflect the change in the file’s input position.
- Restriction:

ts-object must be a [TextStream](#) object file opened with the [OpenTextFile](#) method and with mode `ForReading`.
- See also:

[Assignment](#) statement, [AtEndOfLine](#), [AtEndOfStream](#), [Column](#), [Line](#), [OpenTextFile](#) and [Read](#) methods

TextStream Object

A TextStream object is the name of an open text file.

- 1 **Set** *ts-object* = *fs-object*.**CreateTextFile**(*file-specification*)
- 2 **Set** *ts-object* = *fs-object*.**OpenTextFile**(*file-specification*)
- 3 *ts-object*.*property*
- 4 *ts-object*.*method*

<i>file-specification</i>	»	Expression specifying file name
<i>fs.object</i>	»	Variable name of FileSystemObject object
<i>method</i>	»	Name of TextStream method
<i>property</i>	»	Name of TextStream property
<i>ts-object</i>	»	Variable name of TextStream object

- Operation:**
- Mode 1**—This statement creates a TextStream object for a new file and assigns it to a TextStream variable name.
- Mode 2**—This statement creates a TextStream object for an existing or new file and assigns it to a TextStream variable name.
- Mode 3**—This syntax is used to access one of the properties of an open TextStream object.
- Mode 4**—This syntax is used to perform an operation on an open TextStream object. Operations include reading, writing, skipping and closing.

Notes: A TextStream object is similar to a MultiUser BASIC language file channel number or a C language file handle.

The following properties and methods may be used with TextStream objects:

Properties [AtEndOfLine](#), [AtEndOfStream](#), [Column](#), [Line](#)

Methods [Close](#), [Read](#), ReadAll, ReadLine, [Skip](#), SkipLine, [Write](#), WriteLine, WriteBlankLine.

The *file-specification* is a file name with optional account and path specified. This file name refers to a file in the server system's real file system, not the server's virtual file system. To use a *file-specification* that refers to a file in the virtual file system use the [Server.MapPath](#) object first or as the argument to [CreateTextFile](#) or [OpenTextFile](#)

A special `TextStream` object for the document being created by the ASP is always open and is referred to by the `TextStream` name [Response](#).

Restriction: `TextStream` objects are only created with a [Mode 1](#) or [Mode 2](#) syntax of the [Set](#) statement.

See also: [CreateTextFile](#), [OpenTextFile](#), [Response](#)

Example:

```
Set fs = Server.CreateObject("Scripting.FileSystemObject")
Set file1 = fs.OpenTextFile("my.file", ForReading)
While Not file1.AtEndOfStream
    TextLine = file1.ReadLine
    ...
Wend
file1.Close
```

While Statement

The While statement repeats one or more statements while a specified condition is true.

While *expression*
 statement or text...
Wend

<i>expression</i>	»	Simple variable name or ASP expression
<i>statement</i>	»	Another ASP statement
<i>text</i>	»	HTML tags or text included in output

Operation: *expression* is evaluated and, if it is true (nonzero), the *statement or text* is processed and the loop is repeated. When *expression* is false (zero), loop execution terminates and the statement or text following the Wend key-word is processed.

Notes: Although the operation of this statement is similar to the Do While...Loop statement ([Mode 5](#) of the [Do](#) statement) it is not identical: there is no defined way to exit a While-Wend loop.

See also: [Do](#), [For](#)

Example:

```
Set fs = Server.CreateObject("Scripting.FileSystemObject")
Set file1 = fs.OpenTextFile("my.file",ForAppending)
While Not file1.AtEndOfStream
    TextLine = file1.ReadLine
    ...
Wend
file1.Close
```

Write Methods

The Write methods write text, text lines or just blank lines to a [TextStream](#) object or to the HTML document.

1 *ts-object*.Write(*string*)

2 *ts-object*.WriteBlankLines(*count*)

3 *ts-object*.WriteLine(*string*)

4 **Response**.Write(*string*)

5 **Response**.WriteLine(*string*)

count

» Number of blank lines to write

string

» Text expression to write

ts-object

» Variable name of [TextStream](#) object

- Operation:

Mode 1—The text *string* is written to the file *ts-object*. No carriage return is appended.

Mode 2—*count* number of carriage returns are written to the file *ts-object*.

Mode 3—The text *string* is written to the file *ts-object* and a carriage return is appended.

Mode 4—The text string is written to the HTML document. No carriage return is appended.

Mode 5—The text *string* is written to the HTML document and a carriage return is appended.

Restriction:

The *ts-object* must be the name of a [TextStream](#) object created with the [CreateTextFile](#) or with the [OpenTextFile](#) methods. If [OpenTextFile](#) was used, it must have been opened with ForAppending or ForWriting.

See also:

[CreateTextFile](#), [OpenTextFile](#), [Response](#)

I: CGITEST Command

Because CGI programs are normally invoked by the HTTP server and run as a background task with no console, it can be difficult and awkward to test an application being developed. This command is used for testing CGI applications in the foreground without using the HTTP server.

CGITEST <i>program</i> (<i>options</i>			
<hr/>			
<i>filename</i>	»	Name of file containing name/value lists for <i>option</i>	
<i>options</i>	»	COOKIE= <i>filename</i>	data for cookies
		HTTP= <i>filename</i>	data for HTTP variables
		VAR= <i>filename</i>	data for form data, <i>etc.</i>
<i>program</i>	»	Name of compiled CGI application to test	

- Operation:** The *program* is executed as a CGI application.
- If one or more of the *options* are used, the files specified by those options are used to set the environment variables to the appropriate values, just as if the program were invoked by a web document with the corresponding values set by the user agent (browser) or FORM submissions, *etc.*
- Returns:** No special value is returned by CGITEST except for normal return codes and messages generated by file not found errors, *etc.*
- Any output generated by *program* is sent to stdout, which is the console unless i/o redirection is used.
- Errors:** Normal file errors are detected and reported by CGITEST. Any errors detected by *program* are reported by that program, just as if it were invoked by a web document.
- Notes:** The contents of each of the *filename* files is a list of name/value pairs separated by an equal sign, with one pair defined per line. For instance, the contents of a VAR=*filename* might be:

```
action=Go to Selection
itemselect=1
```

This file is suitable for testing the MAINMENU example program provided with the CGI ToolKit.

Unless the application generates a very small document, you should probably redirect stdout to a disk file so that the results of the program's execution can be more easily examined.

```
>cd /cgi

>cgitest mainmenu (var=mainmenu.formdata > mainmenu.output

>list mainmenu.output

Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<HTML>
...
```

The *filename* specified with the HTTP= option is used to define the HTTP variables used by [FN.CGI.BROWSER\\$ \(capability\\$ \)](#) function or any of the other variables listed in Table 3: “HTTP Header Variables” on page 149 that the program might use.

The *filename* specified with the COOKIE= option is used to define cookie values used by the [FN.CGI.GET.COOKIE\\$ \(name\\$ \)](#) function. If cookie values work properly when tested with this CGITEST program but do not when the application is used by the HTTP server, the cookie values might not have been defined properly on the browser (*e.g.*, the *path\$* or *domain\$* might have been specified incorrectly) or the browser might not support cookies or the user might have them disabled.

The *filename* specified with the VAR= option is used to define values used by the [FN.CGI.GET\\$ \(name\\$ \)](#) or [FN.CGI.GET.FORM\\$ \(name\\$ \)](#) functions.

Glossary of Terms

Active Server Page	ASP. A document containing programming statements interpreted by the server before delivering the document to the client.	client/server architecture	The structure of services that run on the Internet or an intranet. The client computer accesses the server, which supplies the client with resources or information not found on the client's own host.
anonymous logons	This feature allows remote access only by the IUSR_computername account. Remote users can connect to that computer only without a user name and password, and they have only the permissions assigned to that account.	Common Gateway Interface	CGI. An interface used by an application that runs on a Web server when a client requests it.
ASP	See Active Server Page .	connected user	A user who is currently accessing one of the servers.
authentication	A process that confirms that a user is the user specified. Generally done by requiring a password.	data integrity	A way of preventing data from being altered in transit.
bits per second	The measure of speed at which data is transferred over a network.	DHCP	See Dynamic Host Configuration Protocol .
bps	See bits per second .	dial-up	A connection to a computer by telephone, through a modem.
browser	A tool for navigating and accessing information on the Internet or an intranet.	DNS	See Domain Name System .
cache	A store of files from a Web server copied locally for quicker access. To configure your cache on the Internet Explorer browser, from the View menu choose Cache Settings.	domain	A collection of computers that share a common domain database and security policy. Each domain has a unique name.
CGI	See Common Gateway Interface .	Domain Name System	DNS. A protocol and system used throughout the Internet to map Internet Protocol (IP) addresses to names.

Dynamic Host Configuration Protocol	DHCP. An industry-standard protocol that assigns Internet Protocol (IP) configurations to computers.	hyperlink	A way of jumping to another place on the Internet. Hyperlinks usually appear in a different format from regular text.
dynamic document	A document whose content may be different each time that it is retrieved.	hypertext	Documents with links to other documents.
encryption	A way of making data indecipherable while it is being sent from computer to computer.	Hypertext Markup Language	HTML. The formatting language used for documents on the World Wide Web .
File Transfer Protocol	FTP. An industry standard for sharing files between computers.	Hypertext Transfer Protocol	HTTP. The underlying protocol by which WWW clients and servers communicate.
firewall	A system or combination of systems that enforces a boundary between two or more networks and keeps hackers out of private networks.	Integrated Services Digital Network	ISDN. A connection to the Internet installed by your Internet service provider (ISP). A dial-up ISDN line can offer speeds up to 128,000 bps.
FTP	See File Transfer Protocol .	Internet	The global network of computers that communicate through a common protocol, TCP/IP.
gateway	A hardware or software device that directs network traffic.	Internet Protocol	IP. The part of TCP/IP that routes messages from one Internet location to another.
home directory	The root directory for a service. By default, the home directory and all its subdirectories are available to users.	Internet Protocol address	A unique address that identifies a host on a network. It identifies a computer as a 32-bit address that is unique across a TCP/IP network. An IP address is usually represented in dotted-decimal notation, which depicts each octet (eight bits, or one byte) of an IP address as its decimal value
HTML	See Hypertext Markup Language .		
HTTP	See Hypertext Transfer Protocol .		

	and separates each octet with a period, for example: 102.54.94.97.	MIME	<u>M</u> ultipurpose <u>I</u> nternet <u>M</u> ail <u>E</u> xtensions. An extension to Internet e-mail which provides the ability to transfer non-textual data, such as data files. Defined in RFC 2045 - RFC 2049 .
Internet Service Providers	ISP. Public providers of remote connections to the Internet.	MIME mapping	See Multipurpose Internet Mail Extension mapping .
InterNIC	Organization responsible for registering domain names on the Internet.	Multipurpose Internet Mail Extension mapping	MIME. A way of configuring browsers to view files that are in multiple formats.
intranet	A TCP/IP network that can be connected to the Internet but is usually protected by a firewall or other device (for example, a corporate network).	name resolution	A configuration that maps names to IP addresses.
IP	See Internet Protocol .	packet	A piece of information sent over a network.
IP address	See Internet Protocol address .	page	See Web page .
ISDN	See Integrated Services Digital Network .	password authentication	See authentication .
ISPs	See Internet Service Providers .	port number	A number identifying a certain Internet application. For example, the default port number for the WWW service is 80.
leased line	A high-capacity line (most often a telephone line) dedicated to network connections.	protocol	Software that allows computers to communicate over a network. The Internet protocol is TCP/IP.
link	See hyperlink .	proxy	A software program that connects a user to a remote destination through an intermediary gateway.
log file	The file in which logging records are stored.		
logging	Storing information about events that occurred on a firewall or network.		

RFC 959	The definition of the standard used for FTP Clients and servers.	Transmission Control Protocol/Internet Protocol	TCP/IP. A networking protocol that allows computers to communicate across inter-connected networks and the Internet. Every computer on the Internet supports TCP/IP.
RFC 1945	The definition of the HTTP protocol version 1.0.	Uniform Resource Locator	URL. A naming convention that uniquely identifies the location of a computer, directory, or file on the Internet. The URL also specifies the appropriate Internet protocol, such as HTTP.
RFC 2045 - RFC 2049	The definition of the Multipurpose Internet Mail Extensions. <i>See</i> MIME .	URL	<i>See</i> Uniform Resource Locator .
router	A hardware or software device that directs network traffic.	Usenet	The most popular news group hierarchy on the Internet.
script	A group of directives to an application or utility program. A CGI application, for example. <i>See also</i> Common Gateway Interface .	User-agent	Generic term for a network client program such as an FTP client or a Web browser .
Simple Mail Transfer Protocol	SMTP. A protocol used for exchanging mail on the Internet.	virtual directory	A directory outside the home directory that appears to browsers as a subdirectory of the home directory. For any of the servers (WWW or FTP), you can configure a virtual directory through the Setup for the server.
SMTP	<i>See</i> Simple Mail Transfer Protocol .	W3C	World Wide Web Consortium. Founded in 1994 to develop common protocols for the WWW, it is an international consortium hosted by the Massachusetts Institute of Technology Labo-
static document	HTML pages prepared in advance of the request and sent to the client upon request. This page takes no special action when requested. <i>See also</i> dynamic document .		
subnet mask	A TCP/IP configuration parameter that extracts network and host configuration from an IP address.		
TCP/IP	<i>See</i> Transmission Control Protocol/Internet Protocol .		

ratory for Computer Science [MIT/LCS] in the United States, the Institut National de Recherche en Informatique et en Automatique [INRIA] in Europe, and the Keio University Shonan Fujisawa Campus in Asia.

Web browser	A software program, such as Internet Explorer, Netscape Navigator or Netscape Communicator that retrieves a document from a Web server, interprets the HTML codes, and displays the document to the user with as much graphics as the software can supply.
Web page	A World Wide Web document. Pages can contain almost anything, such as news, images, movies, and sounds.
Web server	A computer equipped with the server software to respond to Web client requests, such as requests from a Web browser. A Web server uses the Internet HTTP and FTP protocols to communicate with clients on a TCP/IP network.
World Wide Web	The most graphical service on the Internet. The Web also has the most sophisticated hypertext linking abilities.
WWW	See World Wide Web .

Index

A

- Acquiring an Updated Copy of
 THEO+Server?? 19
- Active Server Page 239
- Active Server Pages
 - See: ASP*
- Allow IP and Deny IP Lists?? 57
- Anonymous
 - See: FTP, user accounts, anonymous*
- Anonymous logons 239
- ASP 239
 - Abs
 - See: ASP, Functions*
 - Browser capabilities 216
 - CBool
 - See: ASP, Functions*
 - CDate
 - See: ASP, Functions*
 - Chr
 - See: ASP, Functions*
 - CInt
 - See: ASP, Functions*
 - Constants 191
 - CStr
 - See: ASP, Functions*
 - Date
 - See: ASP, Functions*
 - Date and time
 - See: ASP, Functions, Date, Day, FormatDateTime, Hour, Minute, Month, Now, Second, Time, WeekDay, Year*
 - Day
 - See: ASP, Functions*
 - Expression operators 193
 - Expressions 191
 - File object properties 195, 232
 - AtEndOfLine 197
 - AtEndOfStream 197
 - Column 199
 - Line 199
 - FormatDateTime
 - See: ASP, Functions*
 - Functions 194, 207
 - Abs 207–208
 - CBool 207–208
 - CDate 207–208
 - Chr 207–208
 - CInt 207, 209
 - CStr 207, 209
 - Date 209
 - Date 207
 - Day 207, 209
 - FormatDateTime 207, 209
 - Directives 212
 - Hour 207, 209
 - InStr 209
 - InStr 207
 - Lcase 207, 210
 - Left 207, 210
 - Len 207, 210
 - Ltrim 207, 210
 - Mid 207, 210
 - Minute 207, 210
 - Month 207, 210
 - Now 210
 - Now 207
 - Right 207, 210
 - Rtrim 207, 210
 - Second 207, 210
 - String 207, 210
 - Time 210
 - Time 207
 - Trim 207, 210
 - Ucase 207, 210
 - WeekDay 207, 211
 - Year 207, 211
 - General Syntax 189
 - Hour
 - See: ASP, Functions*
 - InStr
 - See: ASP, Functions*
 - Lcase
 - See: ASP, Functions*
 - Left
 - See: ASP, Functions*

- Len
 - See:* ASP, Functions
- Ltrim
 - See:* ASP, Functions
- Methods 195, 232
 - Close 198
 - CreateTextFile 200
 - OpenTextFile 218
 - Read 220
 - ReadAll 220
 - ReadLine 220
 - Server.HTMLEncode 228–229
 - Server.MapPath 228–229
 - Server.URLEncode 228–229
 - Skip 231
 - SkipLine 231
 - Write 235
 - WriteBlankLines 235
 - WriteLine 235
- Mid
 - See:* ASP, Functions
- Minute
 - See:* ASP, Functions
- Month
 - See:* ASP, Functions
- Now
 - See:* ASP, Functions
- Objects 195
 - BrowserType 216, 229
 - FileSystemObject 204
 - Request 221
 - Request.Cookies 221
 - Request.Form 221
 - Request.QueryString 221
 - Request.ServerVariables 221
 - Response 224
 - Response.Cookies 224
 - Response.Redirect 224
 - Response.Write 224, 235
 - Response.WriteLine 224, 235
 - Scripting.FileSystemObject 229
 - Server 228
 - Variables 222
 - TextStream 232
- Right
 - See:* ASP, Functions
- Second
 - See:* ASP, Functions
- Statements 189, 194
 - = 196
 - Assignment 196
 - Do 202
 - Do Until... Loop 202
 - Do While... Loop 202
 - Do... Loop 202
 - Do... Loop Until 202
 - Do... Loop While 202
 - Exit Do 202
 - Exit For 205
 - For...Next 205
 - If 214
 - Select Case 226
 - Set 200, 204, 216, 218, 228, **230**, 232
 - While 234
- String
 - See:* ASP, Functions
- Syntax
 - Casemode 190
 - Comments 191
 - White space and carriage returns 190
- Time
 - See:* ASP, Functions
- Trim
 - See:* ASP, Functions
- Ucase
 - See:* ASP, Functions
- Variables
 - Boolean 192
 - Conversion
 - See:* ASP, Functions, CBool, CDate, CInt, CStr
 - Date and Time 192
 - Integer 192
 - Names 192
 - String 193
- WeekDay
 - See:* ASP, Functions
- Year
 - See:* ASP, Functions
- Authentication 239

B

Bits per second

See: bps

blank.gif 106

bps 239

Browser 239

Browser capabilities

See: ASP, Browser capabilities

Browser Identification 121

C

Cache 239

CGI 48, 239

 functions 157

 CGI.BEGIN 159–162

 CGI.ERROR 164–165

 CGI.FINISH 159, 161

 CGI.INIT 159–160, 162

 CGI.LOCATION 164–165

 CGI.PUT.COOKIE 159–161, 163

 CGI.STATUS 164–165

 FN.CGI.BROWSER\$ 166

 FN.CGI.GET\$ 168

 FN.CGI.GET.COOKIE\$ 168–169

 FN.CGI.GET.FORM\$ 168–169

 FN.FORM\$ 171, 173

 FN.FORM.BUTTON\$ 171

 FN.FORM.CHECKBOX\$ 171, 174, 176

 FN.FORM.HIDDEN\$ 171, 174, 176

 FN.FORM.INPUT\$ 171, 174, 176

 FN.FORM.PASSWORD\$ 172, 174

 FN.FORM.RADIO\$ 172, 174, 176

 FN.FORM.RESET\$ 172, 174

 FN.FORM.SELECT\$ 172, 175

 FN.FORM.SELECT.OPTION\$ 172, 175–176

 FN.FORM.SUBMIT\$ 172, 175

 FN.FORM.TEXTAREA\$ 173, 175

 FN.HTML.BREAK\$ 178–179

 FN.HTML.CENTER\$ 178–179

 FN.HTML.DECODE\$ 178–179

 FN.HTML.ENCODER\$ 178–179

 FN.HTML.HEADING\$ 178, 180,

183

 FN.HTML.IMAGE\$ 178, 180

 FN.HTML.LINK\$ 178, 181

 FN.HTML.LIST.ITEM\$ 178, 181

 FN.HTML.MAILTO\$ 178, 181

 FN.HTML.PARA\$ 178, 181

 FN.HTML.RULE\$ 178, 182

 FN.HTML.TAG\$ 178, 182

 FN.JAVA.BUTTON\$ 184

 FN.JAVA.CHECKBOX\$ 184

 FN.JAVA.FORM\$ 184

 FN.JAVA.INPUT\$ 184

 FN.JAVA.PASSWORD\$ 184

 FN.JAVA.RADIO\$ 184

 FN.JAVA.RESET\$ 184

 FN.JAVA.SELECT\$ 184

 FN.JAVA.SUBMIT\$ 184

 FN.JAVA.TEXTAREA\$ 184

 FN.TABLE\$ 185–186

 FN.TABLE.DATA\$ 185–186

 FN.TABLE.HEAD\$ 185–186

 FN.TABLE.RECORD\$ 185–186

 FN.URL.DECODE\$ 188

 FN.URL.ENCODER\$ 188

 limitations 115

 program location 114

CGITEST Command 237

Client/Server architecture 239

Client-Side JavaScript 116

Common Gateway Interface 113, 239

See also: CGI

Connected user 239

Contacting THEOS 127

Controlling Access

 by domain name 93

 by IP Number 93

 by search engines 97

 disallow 99

 to HTTP file system 94

 to specific directories 96

 to the server's file system 59

 to the servers 55

Controlling access

 maximum anonymous users 30

 maximum user attempts 30

 maximum users 30

- to HTTP virtual directory 50–51
- Controlling Access to FTP file system 69
- Cookies 123
 - and ASP 125, 221, 224
 - and CGI 124
 - and SSI 124
 - Getting value of 221
 - See:* CGI, ASP
 - Setting value of 224

D

- Data integrity 239
- Date
 - See:* ASP, Date functions
- Date and time
 - formatted by DATEFORM 141
- DATEFORM environment variable 141
- Day number
 - display 141
 - See:* ASP, Date functions
- DHCP 239
- Dial-up 239
- DNS 239
- Domain 239
- Domain Name System 239
- Dynamic document 240
- Dynamic Host Configuration Protocol 240
- Dynamic Web Pages 109

E

- Encryption 240
- Environment variables
 - CGI startup 93
 - DATEFORM 141
- Extension for
 - SSI documents 46

F

- File association (HTTP) 44, 52
- File Transfer Protocol 240
- Files
 - THEO+Server 129
- Firewall 240
- folder.gif 106

- Form data
 - using 125–126
- Form input, validating (CGI) 117
- Forms
 - Getting fields from 221
- FTP 240
 - directory
 - creating special 21
 - directory change message 32, 82–83
 - E-mail
 - %help FTP variables 31
 - limiting number of connections 68
 - log file 131
 - logging
 - log file directory 32
 - log file frequency 33
 - login message 38, 42, 80
 - lowercase directory name 31
 - message variable
 - %help 31
 - message variables 135
 - messages
 - customizing 29, 34
 - quick setup 21
 - server logging 61
 - commands 34
 - get 34
 - IP names 34
 - put 34
 - replies 34
 - security 33
 - system 33
 - setting up 28
 - access 29, 35
 - General Options 29–30
 - Groups/Users 29, 37
 - log options 29, 32
 - network security 44, 51
 - signoff messages 34, 79
 - signon messages 34, 79
 - sign-off message
 - example 79
 - sign-on message
 - example 79
 - starting 23, 67
 - stopping 67

- TCP/IP port number 30
- user account
 - creating 22
 - defining 74
 - definition 41
 - name 41, 50
- user accounts
 - adding 76
 - anonymous 84
 - check password 31
 - time-out 31
 - group definition 37
 - group name 37, 42
 - groups 37, 74
 - home directory 38, 42, 70
 - maintaining 76
 - password 42
 - time-out 31
- virtual directories
 - access permissions 72
 - hide hidden 38, 42
 - real directory name 39
- virtual directory
 - access permissions
 - create 40
 - delete 40
 - listing 40
 - modify 40
 - read 39
 - subdirectory 40
 - write 39
- FTP Server 9
- FTP server
 - dial-up networking 68

G

Gateway 240

H

- Hardware requirements 11
- Hidden files
 - FTP 38, 42
- Home directory 240
- HTML 240
 - character code encoding (table) 183

- standard color names (table) 161
- HTTP 240
 - customizing common resources 105
 - customizing directory view 102
 - default documents
 - setting up 46, 100
 - directories
 - setup 44, 48
 - directory view 101
 - enabling 47
 - display
 - date
 - day number 141
 - formatted 141
 - month name 141
 - abbreviated 141
 - weekday name 141
 - abbreviated 141
 - time
 - formatted 141
 - error messages
 - custom 139–140
 - customizing standard 106
 - file associations 44, 52
 - header variables (table) 149
 - log file 48, 133
 - logging
 - log file frequency 47
 - quick setup 24
 - server logging 62
 - server name
 - defining 24, 45
 - setting up
 - resources directory 49
 - service options 44–45
 - SSI file extension 46
 - user security 44, 50
 - virtual directories 44
 - setup 28, 44
 - starting 25, 47, 92
 - startup environment variables 93
 - stopping 92
 - TCP/IP port number 45
 - virtual directories
 - real directory name 49
 - virtual directory

- area name 50
 - password 51
 - path 50
- HTTP common resources
 - icons 105
- HTTP server
 - dial-up networking 93
- Hyperlink 240
- Hypertext 240
- Hypertext Markup Language 240
- Hypertext Transfer Protocol 240

I

- Icons (HTTP) 105
- Inactive client, timeout for 31, 46
- Installation procedures 17
- Integrated Services Digital Network 240
- Internet 240
 - access 55
- Internet Protocol 240
- Internet Protocol address 240
- Internet Service Providers 241
- InterNIC 241
- Intranet 241
- IP 241
- IP address 55, 241
- ISDN 241
- ISP 56, 241

J

- JavaScript 116–117
 - disabling with comment tags 118
 - specifying a file as source 118

L

- Leased line 241
- Link 241
- Locale
 - date and time format 141
- Location of
 - directory change messages (FTP) 83
 - login messages (FTP) 81
- Log file 241
 - FTP 131

- HTTP 48, 133
- Logging 241
- Logging user activities 61

M

- Maximum
 - FTP anonymous users 30
 - FTP users 30
 - FTP users probes 30
- MIME 241
- MIME mapping 241
- Month name
 - display 141
 - See:* ASP, Date functions
- Multipurpose Internet Mail Extension map-
ping 241
- MultiUser BASIC language
 - API

N

- Name resolution 241

P

- Packet 241
- Page 241
- parent.gif 106
- Password authentication 241
- Port number 241
- Protocol 241
- Proxy 241

Q

- Query String 122
 - and ASP 123
 - and CGI 122
 - and SSI 122
- Quick setup
 - FTP 21
 - HTTP 24
 - THEO+Server 21

R

RFC 959 242
 RFC 1945 242
 RFC 2045-2049 242
 Router 242

S

Script 242
 SCRIPT tag, using 117
 Server Security 55
 Server Side Includes (SSI) 110
 Server variables
 in ASP 120
 in CGI 119
 in SSI 119
 list of (table) 147
 Setup Command 27
 Setup FTP
 FTP
 setup 29
 Setup HTTP 44
 Setup menu 27
 Sign-off message
 FTP 79
 Sign-on message
 FTP 79
 Simple Mail Transfer Protocol 242
 SLIP 11
 SMTP 242
 Software Requirements 11
 SSI
 syntax 110, 137
 SSI Directives 137
 SSI directives
 #config 139
 TimeFmt Codes 141
 #counter 144
 #echo 147
 #exec 150
 #flastmod 152
 #fsize 154
 #include 155
 #nossi 156
 Starting

FTP server 23
 at boot 67
 FTP server manually 67
 HTTP server 25
 at boot 47, 92
 HTTP server manually 92
 Static document 242
 Stopping
 FTP server 67
 HTTP server 92
 Subnet mask 242
 Support
 Contacting THEOS 127
 technical 127
 Support files
 THEO+Server 129

T

TCP/IP 242
 Technical support 127
 Testing
 FTP server configuration 86
 THEO+Net
 Hardware requirements 11
 Software requirements 11
 THEO+Server
 quick setup 21
 support files 129
 Timeout, inactive client (FTP) 31
 Timeout, inactive client (HTTP) 46
 Transmission Control Protocol/Internet Protocol 242

U

Uniform Resource Locator 242
 unknown.gif 106
 URL 242
 Usenet 242
 User-agent 99, 242
 Users 37
 Using custom messages (FTP) 78
 Using groups 74

V

Virtual directories

60

FTP 39, 71

name 39

HTTP 49, 95

name 49

Virtual directory 242

W

W3C 242

Web

directory view

_DESCRIP. 104

_FOOTING. 103

_HEADING. 102

customizing 102–104

Web browser 243

Web page 243

Web server 243

Weekday name

display 141

See: ASP, Date functions

World Wide Web 243

WWW 243