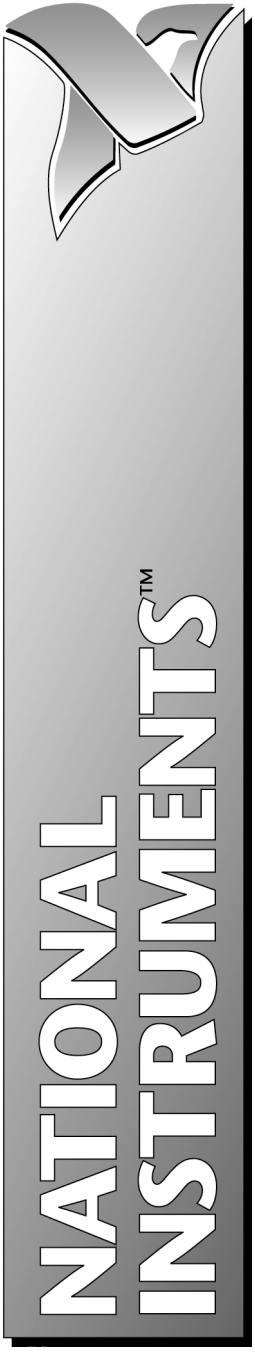

Internet Developers Toolkit for G Reference Manual

January 1997 Edition
Part Number 321392A-01

© Copyright 1997 National Instruments Corporation. All rights reserved.

Internet Support

support@natinst.com
E-mail: info@natinst.com
FTP Site: ftp.natinst.com
Web Address: http://www.natinst.com

Bulletin Board Support

BBS United States: (512) 794-5422
BBS United Kingdom: 01635 551422
BBS France: 01 48 65 15 59

Fax-on-Demand Support

(512) 418-1111

Telephone Support (U.S.)

Tel: (512) 795-8248
Fax: (512) 794-5678

International Offices

Australia 02 9874 4100, Austria 0662 45 79 90 0, Belgium 02 757 00 20,
Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521, Denmark 45 76 26 00,
Finland 09 527 2321, France 01 48 14 24 24, Germany 089 741 31 30, Hong Kong 2645 3186,
Israel 03 5734815, Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456,
Mexico 5 520 2635, Netherlands 0348 433466, Norway 32 84 84 00, Singapore 2265886,
Spain 91 640 0085, Sweden 08 730 49 70, Switzerland 056 200 51 51, Taiwan 02 377 1200,
U.K. 01635 523545

National Instruments Corporate Headquarters

6504 Bridge Point Parkway Austin, TX 78730-5039 Tel: (512) 794-0100

Important Information

Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this manual is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

Trademarks

BridgeVIEW™, LabVIEW®, National Instruments™, and natinst.com™ are trademarks of National Instruments Corporation.

Product and company names listed are trademarks or trade names of their respective companies.

WARNING REGARDING MEDICAL AND CLINICAL USE OF NATIONAL INSTRUMENTS PRODUCTS

National Instruments products are not designed with components and testing intended to ensure a level of reliability suitable for use in treatment and diagnosis of humans. Applications of National Instruments products involving medical or clinical treatment can create a potential for accidental injury caused by product failure, or by errors on the part of the user or application designer. Any use or application of National Instruments products for or involving medical or clinical treatment must be performed by properly trained and qualified medical personnel, and all traditional medical safeguards, equipment, and procedures that are appropriate in the particular situation to prevent serious injury or death should always continue to be used when National Instruments products are being used. National Instruments products are NOT intended to be a substitute for any form of established process, procedure, or equipment used to monitor or safeguard human health and safety in medical or clinical treatment.

Contents

About This Manual

Organization of This Manual xi
Conventions Used in This Manual xii
Related Documentation xiii
Customer Communication xiv

Chapter 1
Introduction

What Is the Internet Developers Toolkit for G? 1-1
How Do I Install My Toolkit? 1-1
Microsoft Windows 3.1 1-1
Microsoft Windows 95 1-2
Microsoft Windows NT 1-2
680x0 Macintosh 1-2
Power Macintosh 1-2
Solaris 1 1-3
Solaris 2 1-3
HP-UX 1-3
Files and Directories 1-4
What Can I Do with the Toolkit? 1-4
Send E-mail and Data Files 1-5
Convert VIs to Internet Applications 1-5
Build CGI Programs in G 1-5
How Do I Configure My Toolkit VIs? 1-6
[ftp] Configuration 1-6
[smtp] Configuration 1-6
[http] Configuration 1-7

Chapter 2
FTP VIs

What Is FTP? 2-1
What Are FTP VIs? 2-1
Low-Level VIs 2-2
Intermediate VIs 2-2
High-Level VIs 2-3
Which FTP VIs Can I Use? 2-3

Chapter 3
E-mail VIs

What Are E-mail VIs? 3-1
Low-Level VIs 3-1
Intermediate VIs 3-2
High-Level VIs 3-2
What Is a Character Set? 3-2
US-ASCII Character Set 3-3
ISO Latin-1 Character Set 3-3
Macintosh Character Set 3-4
Transliteration 3-4
Which E-mail VIs Can I Use? 3-5

Chapter 4
Telnet VIs

What Is a Telnet Connection? 4-1
Network Virtual Terminal 4-1
Negotiated Options 4-2
How Do I Use the Telnet VIs? 4-2
Which Telnet VIs Can I Use? 4-2

Chapter 5
URL VIs

What Is a URL? 5-1
Which URL VIs Can I Use? 5-2

Chapter 6
G Web Server

What Is the G Web Server? 6-1
HTTP Server 6-1
Why Use the G Web Server? 6-3
How Do I Configure My G Web Server? 6-4
Access Configuration 6-5
Configuration Directives 6-6
Server Configuration 6-6
Configuration Directives 6-6
Server Resource Map Configuration 6-7
Configuration Directives 6-8
What Is HTML? 6-8
How Do My VIs Work With the G Web Server? 6-10
View Running VIs 6-10
Execute VIs 6-11
Embed Images 6-12
What URLs Can I Use With My Panel Images? 6-12
Panel Image Formats 6-13
Static Panel Image (.snap URL) 6-13
Syntax 6-13
Examples 6-14
Animated Panel Image (.monitor URL) 6-14
Syntax 6-15
Examples 6-16
Animated Panel Image (.spool URL) 6-16
Syntax 6-17
Examples 6-17
What Is the Common Gateway Interface? 6-18
CGI VIs With the HTTP Server 6-20
What Are CGI Utility VIs? 6-22
CGI Parameter VIs 6-22
File Path VIs 6-23
Network VIs 6-24
Panel Image VIs 6-24
Keyed Array VIs 6-25
HTML VIs 6-26
How Do I Maintain Client State Information? 6-32
Client-Side Cookies 6-33
Server-Side Cookies 6-33

Using Cookies 6-34
Cookie Example 6-34
Cookie VIs 6-34

Appendix A
Configuration Directives

Wildcard Expressions A-1
Access Configuration Directives A-1
Directory A-2
Panel A-2
AllowOverride A-3
AuthName A-4
AuthType A-5
AuthUserFile A-5
AuthGroupFile A-6
Limit A-7
Order A-8
Deny A-9
Allow A-10
Require A-11
Satisfy A-12
OnDeny A-13
Server Configuration Directives A-14
Port A-14
UseDNS A-15
ServerAdmin A-15
ServerRoot A-16
ServerName A-17
StartServers A-17
TimeOut A-18
AccessConfig A-18
ResourceConfig A-19
TypesConfig A-20
CGICacheTime A-20
ErrorLog A-21
TransferLog A-22
AgentLog A-23
RefererLog A-24
LogOptions A-26
PanelImageType A-27
PanelImageDepth A-28
PanelImageQuality A-28

PanelImageCacheCompactTime A-29
ServerPushRefresh A-29
ServerPushLifespan A-30
ServerPushMaxConnections A-31
ServerPushMinTime A-31
Server Resource Map Configuration Directives A-32
DocumentRoot A-32
AccessFileName A-33
Redirect A-34
Alias A-34
ScriptAlias A-35
AddType A-36
AddEncoding A-37
DefaultType A-37
DirectoryIndex A-38
FancyIndexing A-39
DefaultIcon A-39
ReadmeName A-40
HeaderName A-41
AddIcon A-41
AddIconByType A-42
AddIconByEncoding A-43
IndexIgnore A-44
ErrorDocument A-44

Appendix B
Error Codes

Appendix C
Customer Communication

Glossary

Index

Figures

Figure 6-1. HTTP Server in Simple Mode 6-1
Figure 6-2. HTTP Server in Detailed Mode 6-2
Figure 6-3. CGI Application Process 6-19
Figure 6-4. Structure of a CGI Application 6-21

Figure 6-5. HTML Block Diagram 6-26
Figure 6-6. HTML Table Block Diagram 6-27
Figure 6-7. HTML+ String Array to Table VI 6-28
Figure 6-8. HTML+ Monitor VI 6-28
Figure 6-9. HTML Generic Tag VI 6-28
Figure 6-10. HTML Document VI 6-29

Tables

Table 6-1. CGI Environment Variables 6-20

Table B-1. Error Codes B-1

About This Manual

The *Internet Developers Toolkit for G Reference Manual* describes the features, functions, and applications of the Internet Developers Toolkit for G package. You can use this package to send, receive, store, and publish information across the Internet using virtual instruments (VIs) you develop in the G programming environment.

Organization of This Manual

The *Internet Developers Toolkit for G Reference Manual* is organized as follows:

- Chapter 1, *Introduction*, introduces you to the Internet Developers Toolkit for G and describes its features and contents.
- Chapter 2, *FTP VIs*, describes the FTP VIs and their use in the Internet Developers Toolkit for G.
- Chapter 3, *E-mail VIs*, describes the E-mail or SMTP VIs and their use in the Internet Developers Toolkit for G.
- Chapter 4, *Telnet VIs*, describes the Telnet VIs and their use in the Internet Developers Toolkit for G.
- Chapter 5, *URL VIs*, describes the URL VIs and their use in the Internet Developers Toolkit for G.
- Chapter 6, *G Web Server*, describes the G Web Server and how it works with the Internet Developers Toolkit for G.
- Appendix A, *Configuration Directives*, describes the configuration directives for the G Web Server and the use of wildcard expressions within the directives.
- Appendix B, *Error Codes*, lists the error codes returned by the Internet Developers Toolkit for G VIs, including the error number and a description.
- Appendix C, *Customer Communication*, contains forms you can use to request help from National Instruments or to comment on our products and manuals.

- The *Glossary* contains an alphabetical list and description of terms used in this manual, including abbreviations, acronyms, metric prefixes, mnemonics, and symbols.
- The *Index* contains an alphabetical list of key terms and topics in this manual, including the page where you can find each one.

Conventions Used in This Manual

The following conventions are used in this manual:

bold — Bold text denotes a parameter, menu name, palette name, menu item, return value, function panel item, directive, or dialog box button or option.

italic — Italic text denotes emphasis, a cross reference, or an introduction to a key concept.

bold italic — Bold italic text denotes an note.

`monospace` — Text in this font denotes text or characters that you should literally enter from the keyboard. Sections of code, programming examples, and syntax examples also appear in this font. This font also is used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, variables, filenames, and extensions, and for statements and comments taken from program code.

<> — Angle brackets enclose the name of an HTML tag—for example, <B> which tags the text as bold.

» — The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options»Substitute Fonts** directs you to pull down the **File** menu, select the **Page Setup** item, select **Options**, and finally select the **Substitute Fonts** option from the last dialog box.

paths — Paths for URLs and Macintosh and Windows platforms in this manual are denoted using backslashes (\) to separate drive names, directories, and files, as in `C:\dir1name\dir2name\filename`. Paths for UNIX platforms in this manual are denoted using forward slashes (/) to separate drive names, directories, and files, as in `tar xvf/cdrom/solaris1/internet.tar`.

☞ This icon to the left of bold italicized text denotes a note, which alerts you to important information.

Abbreviations, acronyms, metric prefixes, mnemonics, symbols, and terms are listed in the *Glossary*.

Related Documentation

The following documents and World Wide Web sites contain information you might find helpful as you read this manual:

- Chapter 2, *TCP VIs*, in the *LabVIEW Networking Reference Manual* (shipped with LabVIEW 3.1)
- Chapter 2, *TCP VIs*, in the *LabVIEW Communications VI Reference Manual* (shipped with LabVIEW 4.0)
- `http://www.w3.org/pub/WWW/Protocols/` for documentation and specifications of the HTTP
- `http://www.w3.org/pub/WWW/MarkUp/` for documentation and specifications of the HTML
- `http://hoohoo.ncsa.uiuc.edu/cgi/` for documentation and specifications of the CGI
- `http://www.ncsa.uiuc.edu/General/Internet/WWW/HTMLPrimer.html` for a *Beginner's Guide to HTML*

You also can find the following Requests for Comments (RFC) documents at various Web and FTP sites on the Internet. RFC documents form the basis for standard Internet Protocols.

- File Transfer Protocol (FTP)—RFC 959
- HyperText Markup Language—RFCs 1866 and 1942
- HyperText Transfer Protocol (HTTP)—RFC 1945
- Simple Mail Transfer Protocol (SMTP)—RFC 1869
- Telnet Protocol—RFCs 854 and 855
- Multipurpose Internet Mail Extensions (MIME)—RFC 1521

Customer Communication

National Instruments wants to receive your comments on our products and manuals. We are interested in the applications you develop with our products, and we want to help if you have problems with them. To make it easy for you to contact us, this manual contains comment and configuration forms for you to complete. These forms are in Appendix C, *Customer Communication*, at the end of this manual.

Introduction

Chapter 1

This chapter introduces you to the Internet Developers Toolkit for G and describes its features and contents.

What Is the Internet Developers Toolkit for G?

The Internet Developers Toolkit for G provides access to the Internet from within the G environment, using virtual instruments (VIs) and servers. This toolkit consists of four main components:

- FTP VIs—Store and retrieve files from File Transfer Protocol (FTP) servers.
- E-mail VIs—Send messages with attached data to anyone on the Internet.
- Telnet VIs—Send and receive data over the Telnet protocol.
- G Web Server—Publishes your VIs on the World Wide Web (WWW) and remotely controls your VIs from any Web Browser.

How Do I Install My Toolkit?

The following sections contain instructions for installing the Internet Developers Toolkit for G on the Windows, Macintosh, and UNIX platforms.

Microsoft Windows 3.1

Complete the following steps to install the toolkit:

1. Insert the Internet Toolkit CD into your CD-ROM drive.
2. Run the `SETUP` program in the `WIN31\DISK1` directory.
3. The installer asks for a destination directory. Select the directory in which you have BridgeVIEW 1.0 or LabVIEW 4.0 or later installed.

Microsoft Windows 95

Complete the following steps to install the toolkit:

1. Insert the Internet Toolkit CD into your CD-ROM drive.
2. Run the `SETUP` program in the `WIN31\DISK1` directory.
3. The installer asks for a destination directory. Select the directory in which you have BridgeVIEW 1.0 or LabVIEW 4.0 or later installed.

Microsoft Windows NT

Complete the following steps to install the toolkit:

1. Log on to Windows NT as an administrator or as a user with administrator privileges.
2. Insert the Internet Toolkit CD into your CD-ROM drive.
3. Run the `SETUP` program in the `WIN31\DISK1` directory.
4. The installer asks for a destination directory. Select the directory in which you have BridgeVIEW 1.0 or LabVIEW 4.0 or later installed.

680x0 Macintosh

Complete the following steps to install the toolkit:

1. Insert the Internet Toolkit CD into your CD-ROM drive.
2. Run the Installer in the `680x0 Macintosh` folder.
3. The installer asks for a destination directory. Select the directory in which you have LabVIEW 4.0 or later installed.

Power Macintosh

Complete the following steps to install the toolkit:

1. Insert the Internet Toolkit CD into your CD-ROM drive.
2. Run the Installer in the `Power Macintosh` folder.
3. The installer asks for a destination directory. Select the directory in which you have LabVIEW 4.0 or later installed.

Solaris 1

Complete the following steps to install the toolkit:

1. Mount the CD-ROM.
2. Change to the directory in which you have installed LabVIEW. You must have write permission for this directory.
3. Type the following UNIX command:
   `tar xvf /cdrom/solaris1/internet.tar`

Solaris 2

Complete the following steps to install the toolkit:

1. Mount the CD-ROM.
2. Change to the directory in which you have installed LabVIEW. You must have write permission for this directory.
3. Type the following UNIX command:
   `tar xvf /cdrom/solaris2/internet_1/internet.tar`

HP-UX

Complete the following steps to install the toolkit:

1. Mount the CD-ROM.
2. Change to the directory in which you have installed LabVIEW. You must have write permission for this directory.
3. Type the following UNIX command:
   `tar xvf /cdrom/HPUX/INTERNET.TAR`

☞ **Note:** ***The Internet Toolkit has been compiled for LabVIEW 4.0.1, but it also works with BridgeVIEW 1.0 or LabVIEW 4.0 or later. If you install this toolkit into a system other than LabVIEW 4.0.1, you must mass compile the*** `project/internet, user.lib/internet` ***and*** `internet` ***directories before using the Internet Toolkit VIs.***

Files and Directories

After installing the Internet Developers Toolkit for G, the following files and directories are created on your hard drive inside the directory containing your BridgeVIEW or LabVIEW application:

- `project\internet\`—Directory containing the VIs accessible through the **Project»Internet** menu.
- `user.lib\internet\`—Directory containing the VIs accessible through the **User VIs»Internet Toolkit** function and control palettes.
- `internet\`—Directory containing the following configuration files and examples:
  - `internet.ini`—File for configuring your toolkit.
  - `examples\`—Directory containing example VIs for FTP, E-mail, and Telnet.
  - `home\`—Sample home directory containing Common Gateway Interface (CGI) examples for the G Web Server.
  - `smtp\`—Directory containing transliteration files.
  - `http\`—Directory containing the following files for the G Web sever:
    - `conf\`—Directory containing server configuration files.
    - `icons\`—Directory containing image files.
    - `logs\`—Directory containing server log files.

What Can I Do with the Toolkit?

The Internet has created many new opportunities and uses for personal computers and workstations across every industry and application area. Currently, scientists and engineers perform research, publish conclusions, display data, and control source code versions across the Internet.

You can accomplish similar tasks by using your VIs in the G environment. Using the Internet Developers Toolkit for G, you can incorporate the following capabilities into your VI applications:

- Send e-mail or raw data files using your VIs.
- Convert your VIs into Internet-ready applications.
- Build CGI programs in G.

Send E-mail and Data Files

You can add e-mail and FTP transfer capabilities to your VIs written in G with this toolkit. Using these client-side capabilities, you can send e-mail automatically when alarm conditions occur or send files or raw data to an FTP server from your application. You can use these capabilities for any remote field monitoring and control application that must pass data to a centralized reservoir or update operators on the status of the system.

Convert VIs to Internet Applications

You can convert your VIs into Internet-ready applications with this toolkit. You can use the G Web Server to view your VI front panels from any web browser. With the built-in server, you can respond to multiple clients connecting to your program and continuously update the displays for each remote user. If you already are running a server, you can use the toolkit libraries programmatically to convert your VIs into image files for display within HyperText Markup Language (HTML) pages. You also can incorporate security levels into your server to limit access to your front panels and data by controlling access to your VIs based on a user name and password, or based on the user Internet Protocol (IP) address.

Build CGI Programs in G

You can build Common Gateway Interface (CGI) VIs with this toolkit to use with your G Web Server. CGI VIs dynamically make decisions based on user input on a web page. CGI VIs use a standard Application Programming Interface (API) for receiving a request from a remote user and taking action, such as checking an end-user name and password against a valid user list, returning a different HTML page to a user, making a computation and returning the results, or dynamically building an HTML page based on the data passed. This toolkit contains example CGI VIs and a template CGI VI which help you start receiving and sending requests and responses. You can use these examples to build your own CGIs using the G graphical programming language, developed by National Instruments Corporation.

How Do I Configure My Toolkit VIs?

You can leave some parameter settings in the Internet Toolkit VIs unwired. If left unwired, they take the default values configured in the `internet.ini` file, located in the `internet` directory.

The `internet.ini` file contains the following three configuration sections:

- `[ftp]`
- `[smtp]`
- `[http]`

[ftp] Configuration

In the FTP configuration section, specify the following setting for your FTP VIs:

Set the password used to connect to an FTP site.
`AnonymousPassword=email_address`
Example
`AnonymousPassword=joe@some.address.com`

[smtp] Configuration

In the Simple Mail Transfer Protocol (SMTP) configuration section, specify the following settings for your SMTP (E-mail) VIs:

- Set the return address used by outgoing mail.
  `ReturnAddress=email_address`
  Example
  `ReturnAddress=joe@some.address.com <Joe Smith>`
- Set the return subject used by outgoing mail.
  `DefaultSubject=subject`
  Example
  `DefaultSubject=Control System Message`
- Set the server used by the SMTP VIs.
  `Server=internet_smtp_server`
  Example
  `Server=smtp.foo.com`
- Set the character set used by the SMTP VIs.
  `CharSet=character_set`
  Example
  `CharSet=iso-8859-1`

- Set the character mapping settings for converting between virtual and real character sets. The character map settings are a comma-separated list of triplets, three terms listed together in the following form: `<virtual charset> <charset> <mapping file>`.
  `Transliterate=translit_specs`
  Example
  `Transliterate=MacRoman iso-8859-1 macroman.trl`

  This example changes the contents of a text message whose character set is `MacRoman` to a message whose character set is `iso-8859-1` by mapping it with the mapping table stored in the file `macroman.trl`. Refer to the *Transliteration* section in Chapter 3, *E-mail VIs*, for more information.

[http] Configuration

In the HyperText Transfer Protocol (HTTP) configuration section, specify the following setting for your HTTP (G Web) Server:

Set the location of the HTTP Server configuration file.
`lvhttp.cfg=file_path`
Example
`lvhttp.cfg=c:\labview\internet\http\conf\lvhttp.cfg`

☞ **Note:** ***You complete most of the configuration for the G Web Server through separate server configuration files. Refer to Chapter 6, G Web Server, for more information on these specific files.***

FTP VIs

Chapter 2

This chapter describes the FTP VIs and their use in the Internet Developers Toolkit for G.

What Is FTP?

You can use the File Transfer Protocol (FTP) to transfer files between remote machines on the Internet. With FTP, you can download files and directory listings from the server and upload files or create and delete directories on the server. To transfer files to or from a remote host, the host must use FTP server software, and the local machine must use FTP client software to connect to the remote server. Most FTP servers require user authentication to give remote users access to files stored on the server.

When transferring data between an FTP client and an FTP server, you must establish a second connection. You can make this connection either *active* or *passive*. In the default or active mode, the server connects to a data port specified by the client. In the passive mode, the client connects to the server data port. All FTP servers implement the active mode, but it might not work for clients behind fire walls which block all external connection requests. Some older FTP servers might not implement the passive mode, but this mode can work for clients behind fire walls.

What Are FTP VIs?

The FTP VIs implement an FTP client in G. With these VIs, you can transfer files programmatically to and from FTP servers on the Internet in either Image or ASCII (American Standard Code for Information Interchange) mode. When you pass an empty string to the **user name** parameter of an FTP VI, you connect as user `anonymous`. If you pass an empty string for the **password**, the VI uses the `AnonymousPassword` (your e-mail address) specified in the `[ftp]` section of the `internet.ini` configuration file.

You can specify whether data should be transferred in Image or ASCII mode with the VIs that transfer data, such as all of the high-level VIs and some of the intermediate and low-level VIs. You also can determine the way in which the data connection, used for transferring data, is established.

When you transfer text files, use ASCII mode. In this mode, the FTP VIs convert line terminating characters to and from the Internet standard `CRLF`. Use Image mode for binary files, such as when transferring data log files or VIs. In Image mode, the transferred data is not modified.

The FTP VIs are divided into three categories:

- Low-level VIs—Implement the commands in the FTP specification.
- Intermediate VIs—Perform FTP tasks consisting of low-level FTP commands.
- High-level VIs—Perform file transfer tasks when the file names are specified.

Low-Level VIs

You can use the low-level VIs if you want to customize FTP for a specific task, or if you want to use some of the other FTP commands. The low-level VIs directly implement the commands in the FTP specification, such as creating, listing, or deleting directories.

Intermediate VIs

You also can use the intermediate VIs if you want to customize FTP for a specific task, or if you want to use some of the other FTP commands.The intermediate-level VIs are built on top of the low-level VIs and implement common tasks consisting of a series of low-level commands. The intermediate level VIs also include VIs that open and close connections to FTP servers.

High-Level VIs

You can use the high-level VIs to complete entire tasks for you. First, you specify the path on the remote FTP server, along with the user authentication and local file information. Then, the VIs connect to the remote host, transfer data between your computer and the FTP server, and close the connection. With the high-level VIs, you can transfer one or more files directly to and from either memory or files on disk.

Each high-level VI uses the address of the FTP server, a user name and password, remote file paths (paths to location on the server), and local file paths or data (information about where the local data should be) for the input.

Which FTP VIs Can I Use?

You can use the VIs listed below to transfer files between a remote FTP server and a local machine. With these VIs, you can read data from a file, store data to a file, or pass the data as a string parameter (buffer). You can find these VIs in the **User Libraries»Internet Toolkit»FTP VIs** palette. To access more detailed information about these VIs, right-click on the VI and select **Online Help** from the VI pop-up menu.

The following VIs are high-level VIs. Refer to the Online Help file for more information on the low-level and intermediate VIs.

- FTP Get Buffer VI—Connects to an FTP server and returns the contents of a single file.
- FTP Get File VI—Connects to an FTP server and copies a single file to the local machine.
- FTP Get Multiple Buffers VI—Connects to an FTP server and returns the contents of a list of files.
- FTP Get Multiple Files VI—Connects to an FTP server and copies a list of files to the local machine.
- FTP Get Multiple Files and Buffers VI—Connects to an FTP server and copies a list of files, saving them on the local machine or returning the contents.
- FTP Put Buffer VI—Connects to an FTP server and stores the contents of a single buffer.
- FTP Put File VI—Connects to an FTP server and copies the contents of a single file from the local machine.

- FTP Put Multiple Buffers VI—Connects to an FTP server and stores the contents of multiple buffers.
- FTP Put Multiple Files VI—Connects to an FTP server and copies the contents of multiple files from the local machine.
- FTP Put Multiple Files and Buffers VI—Connects to an FTP server and copies the contents of multiple files or buffers from the local machine.

The default settings for the FTP VIs are configured in the `internet/internet.ini` file.

E-mail VIs

Chapter 3

This chapter describes the E-mail or SMTP VIs and their use in the Internet Developers Toolkit for G.

What Are E-mail VIs?

You can use the E-mail VIs to send electronic mail, including attached data and files, using the Simple Mail Transfer Protocol (SMTP). When you use the E-mail VIs, your messages are encoded using the Multipurpose Internet Mail Extensions (MIME) format. In this format, you can send multiple documents, including binary data files, within an e-mail message. You also can describe properties of the individual attachments, for example, what character set a text document uses.

To send e-mail on the Internet, you must have the e-mail address of one or more recipients and the Internet address of an SMTP server. The E-mail VIs open a connection to the server and send the server commands describing the recipients and the contents of the e-mail message. The server sends the message to the individual recipients or forwards it to other SMTP servers.

The E-mail VIs are divided into three categories:

- Low-level VIs—Implement the individual SMTP commands.
- Intermediate VIs—Perform tailored e-mail commands.
- High-level VIs—Send your completed messages for you.

Low-Level VIs

You can use the low-level VIs to access the actual commands defined by the SMTP protocol. Using these VIs, you can circumvent the formatting done by the intermediate VIs, or use commands that are not part of SMTP but that work with your server.

Intermediate VIs

You can use the intermediate VIs to tailor the contents of an e-mail message, as well as to send multiple e-mail messages over one connection. These VIs can open and close connections to an SMTP server and encode message parts to MIME format. The intermediate VIs are built on top of the low-level VIs.

You can leave several input parameters in the intermediate VIs unwired. If left unwired, they use the default values specified in the `internet.ini` configuration file. Refer to the section *[smtp] Configuration*, in Chapter 1, *Introduction*, for more information on these parameters.

High-Level VIs

You can use the high-level VIs to specify the sender, recipients, message, and attachments to your e-mail. These VIs open a connection to an SMTP server, send the message, encode it in MIME format, and close the connection. The high-level VIs are easy to use and cover many e-mail tasks.

Each high-level VI uses the addresses of the recipients, a subject, a message, and attachments as the input. You also can specify the sender, the SMTP server to use, and the character set the message and text attachments use. The high-level VIs are built on top of the intermediate VIs.

You can leave several input parameters in the high-level VIs unwired. If left unwired, they use the default values specified in the `internet.ini` configuration file. Refer to the section *[smtp] Configuration*, in Chapter 1, *Introduction*, for more information on these parameters.

What Is a Character Set?

The high-level VIs and some of the intermediate VIs contain an input parameter called **character set**. This parameter specifies the character set used in the text of the e-mail message or attachment. A **character set** describes the mapping between *characters* and their *character codes*.

A character is a basic unit of written languages; a letter, number, punctuation mark, or in some languages, an entire word. Modifications of letters, such as capitalization or accent marks, make that letter a

separate character. For example, the characters O, o, ö, and ô are all different characters.

A character code is a number used to represent a given character. Because computers only deal with numbers, they must associate a character with a number to operate on the character.

A **character set** is the relationship between characters and the numbers that represent them in the computer. For example, in ASCII, the character codes for A, B, and C, are 65, 66, and 67, respectively.

US-ASCII Character Set

The most widely used character set on the Internet is the *US-ASCII* or *ASCII* character set. Most Internet programs, including e-mail applications, use this character set as the default and do not work with any other sets. The ASCII character set describes all the letters and most punctuation marks used in the English language, for a total of 128 characters. Most other character sets are extensions of ASCII.

Using the ASCII character set might not be sufficient because many languages require characters not found in ASCII. For example, you cannot write the German word Müller using ASCII, because the character ü is not defined in the ASCII character set.

ISO Latin-1 Character Set

Because many languages require characters not found in ASCII, countries using these languages created new character sets. Most of these character sets contain the first 128 character codes as ASCII, but also defined the next 128 character codes to describe characters needed in that language. Some of these character sets describe the same characters but use different character codes for a particular character. This can cause problems when text written in one character set is displayed using another character set. To solve this problem, some standard character sets are used. One widely used character set is *ISO Latin-1*, also known as *ISO-8859-1*. This character set describes characters used in most western European languages and is understood by most e-mail application that deal with these languages.

Macintosh Character Set

Macintosh developed its own extended character set before ISO Latin-1 was defined. The *Macintosh* character set is based on ASCII but with a different set of upper 128 character codes than ISO Latin-1. Because of this, e-mail containing accented characters written in the Macintosh character set appears incorrectly in e-mail applications expecting ISO Latin-1 text. To solve this problem, e-mail applications on the Macintosh convert text to ISO Latin-1 before mailing. When another Macintosh e-mail application receives text designated as using the ISO Latin-1 character set, it converts the message to the Macintosh character set.

Transliteration

Using the E-mail VIs, you can specify character sets that map text to another character set before sending the text. The mapping of characters is called *transliteration*. In the `[smtp]` section of the `internet.ini` configuration file, you can specify which transliterations the E-mail VIs can work with. A transliteration is defined by a *virtual character set*, a *target character set*, and a *transliteration file*. The default configuration file contains one defined transliteration, `[MacRoman iso-8859-1 macroman.trl]`. When you send a message and specify that the character set is `MacRoman`, the E-mail VIs transliterate the text and send it out as character set `iso-8859-1` (ISO Latin-1).

The transliteration file is a binary file of 256 bytes. The value of each entry contains the new character code of the mapped character. Entries with a value the same as their index do not modify that particular character. That is, the identity transliteration file is a file consisting of values 0, 1, 2, 3, …, 255; it does not change the transliterated text. For example, a transliteration file that changes ASCII letters to upper case is the same as the identity transliteration file, except that entries representing lowercase characters (97, 98, …, 122) contain the character codes of the equivalent upper case characters (65, 66, …, 90). If the name of this file is `asciiup.trl`, the transliteration entry in the configuration file might be `ASCIIUpper us-ascii asciiup.trl`.

When a transliteration specifies another transliteration as the target character set, the mappings are applied in the expected order. For example, if the transliteration entry is [`MacRoman iso-8859-1 macroman.trl, MacRomanUp MacRoman asciiup.trl`], the character set `MacRomanUp` first changes all ASCII characters in the text to upper case (using `asciiup.trl`) and then changes the text to ISO Latin-1 (using `macroman.trl`).

Which E-mail VIs Can I Use?

You can use the VIs listed below to send e-mail with zero or more attachments to one or more users. The attachments can come directly from a string or can be a file on a local disk. You can find these VIs in the **User Libraries»Internet Toolkit»Email VIs** palette. To access more detailed information about these VIs, right-click on the VI and select **Online Help** from the VI pop-up menu.

The following VIs are high-level VIs. Refer to the Online Help file for more information on the low-level and intermediate VIs.

- SMTP Send Message VI—Sends text e-mail message to a list or recipients.
- SMTP Send Data VI—Sends an e-mail message with attached data to a list of recipients.
- SMTP Send File VI—Sends an e-mail message with an attached file to a list of recipients.
- SMTP Send Multiple Attachments VI—Sends an e-mail message with multiple data and file attachments to a list of recipients.
- SMTP Send Message (Small) VI—Sends a text e-mail message to a single recipient.

The default settings for the SMTP (E-mail) VIs are configured in the `internet/internet.ini` file.

Telnet VIs

Chapter 4

This chapter describes the Telnet VIs and their use in the Internet Developers Toolkit for G.

What Is a Telnet Connection?

A Telnet connection is a Transmission Control Protocol (TCP) connection used to transmit data containing Telnet control information. The Telnet Protocol provides a general, bi-directional, 8-bit byte-oriented communications facility. The Telnet Protocol also provides a standard method of interfacing terminal devices and terminal-oriented processes to each other.

The Telnet Protocol is built upon the following two ideas:

- the concept of a *Network Virtual Terminal*
- the principle of negotiated options

Network Virtual Terminal

When a Telnet connection is first established, each end is assumed to originate and terminate at a Network Virtual Terminal (NVT). An NVT is an imaginary device which provides a standard, network-wide, intermediate representation of a canonical terminal. An NVT eliminates the need for server and user hosts to keep information about the characteristics of each terminal and terminal handling conventions. All hosts, both server and user, map their local device characteristics and conventions so they appear to be dealing with an NVT over the network, and each host assumes a similar mapping by the other party.

Negotiated Options

The principle of negotiated options recognizes the fact that many hosts want to provide additional services over and above those available within an NVT, and that many users have sophisticated terminals and want elaborate, rather than minimal, services. To meet this need, the Telnet Protocol provides various options that allow a server and user to agree on a set of conventions for their Telnet connection. These options are independent of, but structured within, the Telnet Protocol.

How Do I Use the Telnet VIs?

The Telnet VIs form a transparent layer between your application and the Telnet Protocol. These VIs implement the most basic NVT by interpreting and rejecting all option negotiation requests. With the Telnet VIs, your applications can interface with remote applications using the Telnet Protocol as if it were a regular TCP connection. You can use the Telnet VIs to log on programmatically to a remote shell and execute commands, or to control instruments that use the Telnet Protocol.

Which Telnet VIs Can I Use?

You can use the Telnet VIs listed below to connect your application with the Telnet Protocol using the basic NVTs. You can find these VIs in the **User Libraries»Internet Toolkit»Telnet VIs** palette. To access more detailed information about these VIs, right-click on the VI and select **Online Help** from the VI pop-up menu.

- Telnet Open Connection VI—Attempts to open a Telnet connection with the specified address and port.
- Telnet Read VI—Reads data from the Telnet connection, filtering out Telnet control characters.
- Telnet Write VI—Writes the string data to the specified Telnet connection.
- Telnet Play Script VI—Executes script on the specified Telnet connection.

- Telnet Close Connection VI—Closes the Telnet connection.
- Telnet Listen VI—Creates a listener and waits for an accepted Telnet connection at the specified port.
- Telnet Wait On Listener VI—Waits for an accepted Telnet connection using the specified listener.

URL VIs

Chapter 5

This chapter describes the URL VIs and their use in the Internet Developers Toolkit for G.

What Is a URL?

The Universal Resource Locators (URLs) are the standard method for describing resources on the Internet. A fully formed Internet URL consists of the following format:

```
protocol://user:password@host:port/virtual-path
```

You can omit the authentication (`user:password@`) and the Transmission Control Protocol/Internet Protocol (TCP/IP) port specification (`:port`), which instructs the browser to supply the user name and password and the default port for the specified protocol. Some protocols, such as `mailto`, do not use a virtual path but instead require a user name.

The following list provides some example URLs and their functions.

- `http://www.natinst.com`—Uses the `http` protocol to access the default document from `www.natinst.com`.
- `http://www.natinst.com/labview`—Uses the `http` protocol to access the `/labview` document from `www.natinst.com`.
- `http://some.comp.adr:8080/info.htm`—Uses the `http` protocol to connect to port `8080` at `some.comp.adr` and retrieve the document `/info.htm`.
- `ftp://john:smith@ftp.some.addr/pub/readme.txt`—Uses the `ftp` protocol to log on to `ftp.some.addr` as user `john` with password `smith` and retrieve the file `pub/readme.txt`.
- `mailto:g.internet@natinst.com`—Specifies the e-mail address `g.internet@natinst.com`.

Which URL VIs Can I Use?

You can use the VIs listed below to build and parse URLs and to retrieve documents by specifying their URL. Some of the URL VIs also work with URLs that are not fully specified. You can find these VIs in the **User Libraries»Internet Toolkit»URL VIs** palette. To access more detailed information about these VIs, right-click on the VI and select **Online Help** from the VI pop-up menu.

- Build URL VI—Constructs a URL string from a protocol, host, user password, port, and virtual path.
- Parse URL VI—Parses a fully or partially formed URL and returns its components.
- URL Get HTTP Document VI—Retrieves a document specified by a URL, using the HTTP protocol and following redirections to other HTTP URLs.
- URL Get Gopher Document VI—Retrieves a document specified by a URL using the Gopher protocol.
- URL Get FTP Document VI—Retrieves a document specified by a URL using the FTP protocol.
- URL Get Document VI—Retrieves a document specified by a URL using the HTTP, Gopher, or FTP protocol as specified in the URL and following redirections to other URLs.

G Web Server

Chapter 6

This chapter describes the G Web Server and how it works with the Internet Developers Toolkit for G.

What Is the G Web Server?

The G Web Server is an HTTP/1.0 compatible server for making HTML and other documents available on the Internet and for connecting VIs to the World Wide Web. With the G Web Server, you can publish your VIs on the World Wide Web and control your VIs from any web browser.

The G Web Server, a type of HTTP Server, is a stand-alone VI that executes independently of other VIs that are running. You can start the server by selecting **Project»Internet»Start HTTP Server…**, or by using the HTTP Server Control VI. The HTTP Server Control VI programmatically tests the state of the G Web Server, as well as loads, starts or stops, and unloads the server VIs. To access more detailed information about this VI, right-click on the VI and select **Online Help** from the VI pop-up menu.

HTTP Server

The front panel of the HTTP (G Web) Server displays the server status information. You can view the panel in a *simple mode* or *detailed mode*. Simple mode uses less screen space and is more efficient. Figure 6-1 shows the G Web Server in simple mode.

Figure 6-1. HTTP Server in Simple Mode

The text on the left side of the black bar displays the current date, time, and time zone offset.

The text on the right side of the black bar displays the elapsed time since you started the server. In Figure 6-1, *HTTP Server in Simple Mode*, the server started five days, 17 hours, 27 minutes and 50 seconds ago. It also displays the number of requests the server handled during this time (1544 requests).

The **Stop** button stops the HTTP Server, closes all open connections, and unloads cached CGI VIs from memory.

The bold text displays the current status of the server. You can click on the black triangle to change the panel to detailed mode.

The detailed mode, shown in Figure 6-2, displays information about the current state of connections, current panel animations, active or cached CGI VIs, and a current log of client requests.

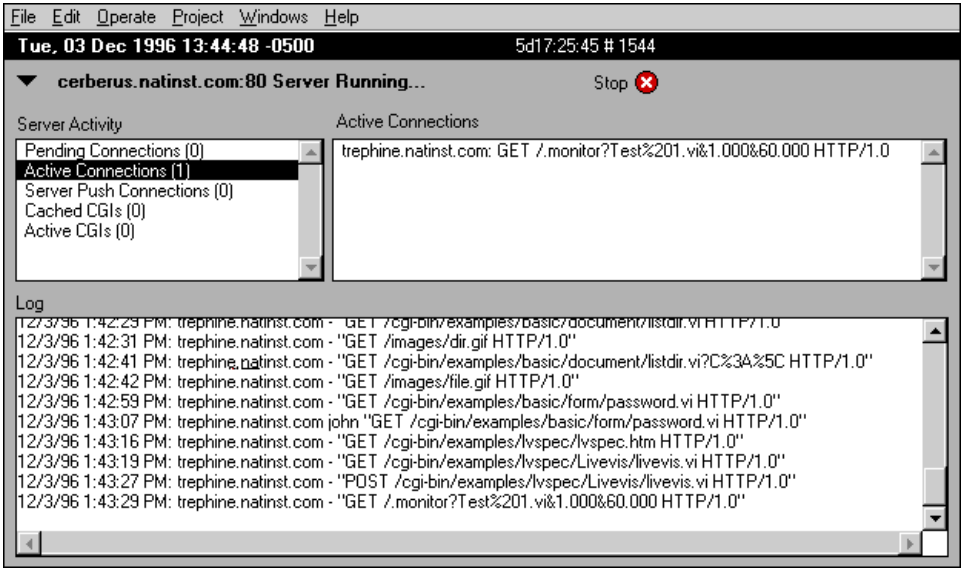

Figure 6-2. HTTP Server in Detailed Mode

In detailed mode, the front panel of the HTTP Server displays the same information as in the simple mode, plus the following information:

- **Server Activity**—Displays information about different server aspects. When you select an item in the list, the box to the right displays more detailed information. You can choose from the following list of items:
  - **Pending Connections**—Indicates the number of connections the server has accepted but not yet processed. Select this item to display the addresses of these connections.
  - **Active Connections**—Indicates the number of all open connections. Select this item to display the addresses and requests of these connections.
  - **Server Push Connections**—Indicates the number of current server-push (image animation) connections. Select this item to display addresses and requests of all server-push connections.
  - **Cached CGIs**—Indicates the number of CGI VIs currently in memory. Select this item to list the CGI names, their activity status, and the number of pending requests for each CGI.
  - **Active CGIs**—Indicates the number of CGI VIs currently processing requests.
- **Log**—Displays the most recent server requests and server error messages. Each request line consist of the date, time, remote machine address, user name (or -), and the request as sent by the client. Error lines depend on the error condition, such as an invalid URL. You can find more complete log information in the server log files.

Why Use the G Web Server?

The G Web Server can perform the following tasks with your VIs.

- Publishes Virtual Instruments on the Web—You can publish static or animated images of your VIs on the Word Wide Web, without modifying your VIs.
- Works with CGI Virtual Instruments—You can develop CGI VIs that execute dynamically when requested by a browser. You can use your CGI VIs with all platforms that work with G.
- Works with CGI animations—You can work with server-push animations generated by CGI VIs.

- Works with Imagemap files—You can work with Imagemap files without an external CGI application.
- Integrates with other systems—You can run the G Web Server on a development system or integrate it into your executable.
- Works on different platforms—You can run the G Web Server on all platforms that work with BridgeVIEW 1.0 or LabVIEW 4.0 or later. CGI VIs written on any of these platforms run on any of the other platforms.
- Implements security—You can limit access by directory or VI name. You can control access by the client address, the referring document, or through group and password files.

How Do I Configure My G Web Server?

When you configure your G Web Server, you edit the server configuration files to specify how the server behaves on your system.

The following rules apply to all of the G Web Server configuration files.

- Case insensitive entries—Configuration entries are not case sensitive, except where path names on case sensitive file systems are involved.
- Comment lines beginning with #—Lines to be ignored must begin with #.
- One directive per line—Each line consists of:

  `Directive data [data2 ... data`*n*`]`

  – `Directive`—A keyword the server recognizes, followed by whitespace. Refer to Appendix A, *Configuration Directives*, for detailed information on each directive.

  – `data`—Specific to each directive. Additional data entries are separated by whitespace.
- Extra whitespace ignored—Embedded spaces between `Directive` and `data` are ignored. To embed a space in the data without separating it from any subsequent arguments, use a backslash (`\`) character before the space.

- File Paths in UNIX format—The path separator is `/`. All absolute paths must begin with this character.
  - On Windows, the following code sets **PathDirective** to the path `C:/labview/web`.
    `PathDirective /c/labview/web`
  - On the MacOS, the following code sets **PathDirective** to the path `Macintosh HD:LabVIEW`. Note that the space is preceded by a \.
    `PathDirective /Macintosh\ HD/LabVIEW`

You must edit three main configuration files for your G Web Server:

- Access Configuration—Controls client access to certain directories, and determines which features are available in different directories.
- Server Configuration—Controls the technical aspects of server operation.
- Server Resource Map Configuration—Controls document and script locations and aliases.

Access Configuration

The Access Configuration controls the following aspects of document directories in a tree with the server:

- Access—Restricts access to a branch of the directory tree to allowed hosts and/or authenticated users.
- Server Features—Secures directories by disabling certain server functions.

You can use two files to control access to directories:

- Global Access Configuration file—A document in your server `conf` directory, specified by the Server Configuration directive **AccessConfig**, that controls access to any directory in your tree.
- Per-Directory Access Configuration file—A file, within the document tree and with the name specified by the Server Resource Map Configuration directive **AccessFileName**, that controls access to the file directory as well as any subdirectories.

Configuration Directives

In addition to the general configuration rules that apply to all HTTP configuration files for G, the access control files also work with directives and sectioning directives.

You can use the directives and sectioning directives listed below to write an Access Configuration file (ACF). Refer to Appendix A, *Configuration Directives*, for detailed information on each directive.

- Directory
- Panel
- AllowOverride
- AuthName
- AuthType
- AuthUserFile
- AuthGroupFile
- Limit
  - Order
  - Deny
  - Allow
  - Require
  - Satisfy
  - Referer
  - OnDeny

Server Configuration

The Server Configuration controls the operation of the server. Use the Server Configuration file, `lvhttp.cfg`, located in the `conf` directory, to complete the configuration.

Configuration Directives

You can use the directives listed below to configure your Server Configuration file. Refer to Appendix A, *Configuration Directives*, for detailed information on each directive.

- Port
- UseDNS

- ServerAdmin
- ServerRoot
- ServerName
- StartServers
- TimeOut
- AccessConfig
- ResourceConfig
- TypesConfig
- CGICacheTime
- Logging
  - ErrorLog
  - TransferLog
  - AgentLog
  - RefererLog
  - LogOptions
- Panel Images
  - PanelImageType
  - PanelImageDepth
  - PanelImageQuality
  - PanelImageCacheCompactTime
- Server Push Animations
  - ServerPushRefresh
  - ServerPushLifespan
  - ServerPushMaxConnections
  - ServerPushMinTime

Server Resource Map Configuration

The Server Resource Map Configuration controls the presentation of the server documents. Use the Server Resource Map file `srm.cfg`, located in the `conf` directory, to complete the configuration.

Configuration Directives

You can use the directives listed below to configure your Server Resource Map file. Refer to Appendix A, *Configuration Directives*, for detailed information on each directive.

- DocumentRoot
- AccessFileName
- Virtual URLs
  - Redirect
  - Alias
  - ScriptAlias
- AddType
- AddEncoding
- DefaultType
- Indexing Options
  - DirectoryIndex
  - FancyIndexing
  - DefaultIcon
  - ReadmeName
  - HeaderName
  - AddIcon
  - AddIconByType
  - AddIconByEncoding
  - IndexIgnore
- ErrorDocument

What Is HTML?

HyperText Markup Language (HTML) is a markup language for hypertext used in World Wide Web clients. HTML documents consist of plain text with embedded tags. You use tags for sectioning HTML documents, formatting the text, setting styles and colors, embedding images and objects in the text, and specifying links to other documents.

Each tag has a unique name enclosed in brackets (<>). Most tags also have a closing tag (</>). For example, if you want to display the following text:

This is a **bold** word.

You use the following string, using the <B> (bold) tag for the opening tag and </B> for the closing tag:

```
This is a <B>bold</B> word.
```

Some HTML tags use optional additional attributes. For example, the paragraph tag <P> can use the ALIGNMENT attribute to specify the horizontal alignment of the paragraph. If you do not specify this attribute, the alignment of the paragraph does not change.

```
<P ALIGN=CENTER>This paragraph is centered</P>
```

If plain text in an HTML document contains characters reserved by HTML, namely <, >, ", and &, you must quote it according to HTML conventions. For example, the string 3<4 would be written as 3<4 in HTML.

The HTML specification continually changes. Because of this, some companies define specific tags only understood by their browser application, but not part of the official standard. This practice usually does not cause problems because the HTML specification states that browsers should ignore tags they do not understand. However, documents using software-specific tags appear incorrectly in other browsers.

The World Wide Web Consortium (W3C), founded to develop common standards for the World Wide Web, keeps track of versions of HTML. As of November 5, 1996, HTML 3.2 is the W3C recommendation. For more information about the state of HTML, refer to the W3C home page at the following address:

```
<http://www.w3.org>
```

You can design HTML documents using an HTML editor or, if you know the syntax of the language, directly in any text editor. The Internet Developers Toolkit for G provides a library of HTML VIs you can use to construct HTML documents programmatically. To use these VIs effectively, learn more about HTML and the different tags it uses. There are many HTML tutorials available on the Internet, and most bookstores carry a good selection of books on HTML. Refer to the *Related*

Documentation section in the chapter *About This Manual* for more information on HTML documents.

How Do My VIs Work With the G Web Server?

You can use the G Web Server to passively monitor the front panels of running VIs, to actively execute CGI VIs on behalf of a remote user request, or embed images in HTML documents.

View Running VIs

You can use the G Web Server to publish the front panel image of any VI in memory. To view the static or animated image of a front panel, an HTML page must contain an embedded image tag whose source URL references the G Web Server and specifies the VI name and, optionally, the image parameters.

For static images, the URL takes the following form:

```
<http://web.server.addr/.snap?VI_Name>
```

The character `?` separates the URL from the parameters. `VI_Name` specifies the name of the returned VI front panel image. You must encode the `VI_Name` according to URL naming rules; that is, replace special characters with their hexadecimal value preceded by a percent (%) sign and replaces spaces with a plus (+) sign. Following the VI name, you can add parameters that specify the image format to use (JPEG or PNG), the pixel depth to return (1,4,8, or 24 bits), and any additional image parameters such as image quality for JPEG and compression level for PNG.

For example, you can write the URL for the static image of the VI `Test Example.vi`, in PNG format and in black and white, as follows:

```
<http://web.server.addr/.snap?Test+Example.vi
&type=png&depth=1>
```

For animated images, the URL takes the following form:

```
<http://web.server.addr/.monitor?VI_Name>
```

The G Web Server uses the server-push method to implement animations of panel images. During a server push, the server maintains an open connection and sends a new image after a predefined period of time. Because only a few browsers currently work with server-push animation, browsers that do not work with it receive a single static

image instead. You can use all the optional parameters for static images with animated images. You also can specify the refresh rate and the length of an animation.

For example, you can write the URL for the animated image of the VI `Test Example.vi`, which updates once every two seconds for three minutes, as follows:

```
<http://web.server.addr/.monitor?Test+Example.vi
&refresh=2&lifespan=180>
```

Execute VIs

You can use the G Web Server to load and execute specially designed VIs dynamically. In order for the server to recognize a VI as a CGI application, you must designate the directory the VI resides in as a CGI directory using the **ScriptAlias** directive. The server uses the CGI to exchange parameters and data with these CGI VIs. You use CGI applications dynamically to create documents whose content frequently changes and to process queries and form requests. Examples of CGI applications include stock market information, fill-out registration forms and online stores. In the G environment, you can use a CGI application to start or change parameters of an experiment written in G. CGI applications execute on the server machine and are different from applets which you download and execute on the machine on which the browser is running.

You can invoke a CGI application from a browser through a link or by clicking on the **Submit** button of an HTML form. The URL for a CGI application appears the same as other documents, except you often specify additional parameters. The server determines whether a URL describes a regular document or a CGI application.

For example, you can write the URL to execute the CGI VI `test.vi` in the `cgi-bin` directory which includes the two parameters **name** and **age** with values `Bob` and `32`, respectively, as follows:

```
<http://web.server.addr/cgi-bin/test.vi?name=
Bob&age=32>
```

You also can create server-push animations on the G Web Server using CGI VIs. You create a special URL instructing the server to execute the CGI server-push animation. The server then repeatedly calls the CGI when a new image frame is needed. In server push animations, the CGI must return images instead of HTML documents. You can use the

refresh and **lifespan** parameters in the same way as in front panel animations.

For example, you can write a URL for a CGI animation using the CGI VI `animate.vi` in the `cgi-bin` directory, which includes the parameter **name** with the value `Bob`, updating once every 5 seconds for a minute, as follows:

```
<http://web.server.addr/.spool?/cgi-bin/animate.vi
&name=Bob&refresh=2&lifespan=60>
```

Embed Images

You must use the Image (`IMG`) tag and specify the location of the image in the source attribute to embed an image in an HTML document. If the image and the document that contains the image are on the same server, you can use the relative path to the image. If the image and the document are on different servers, you must specify the entire URL.

For example, to embed the image `http://foo/images/sample.gif` in a document on the machine `foo`, use the following HTML code:

```
<IMG SRC="/images/sample.gif">
```

Or, to embed the same image in a document not on the machine `foo`, use the following HTML code:

```
<IMG SRC="http://foo/images/sample.gif">
```

The following HTML code describes two headings and two embedded images: a static picture of the VI `test.vi` and an animation of the same VI.

```
<H2>Static Panel Image</H2>
<IMG SRC="/.snap?test.vi" >
<H2>Animated Panel Image</H2>
<IMG SRC="/.monitor?test.vi">
```

What URLs Can I Use With My Panel Images?

With the G Web Server, you can publish images of your VI panels on the World Wide Web. You do not need to modify the VIs in order to display their panels.

Panel Image Formats

The G Web Server can generate images of VI front panels in the Joint Photographic Experts Group (JPEG) and Portable Network Graphics (PNG) image formats.

The JPEG image format is a public domain image format supported by all current browsers. It has been developed for the distribution of real-life images and photographs and uses a lossy compression algorithm for reducing the memory size of an image. When you use JPEG on images containing lines and text, such as front panels, the resulting image often displays artifacts of the compression, such as fuzzy text or stray color pixels.

The PNG format is a brand new public domain image format. The compression algorithm in this format is lossless, which produces PNG images exactly like the original images. PNG is designed to be the successor of the GIF format, which also uses lossless compression. PNG is an open standards that you also can use on true color images. However, because the format is new, most browsers currently require a plug-in or external application to view PNG images. This will become unnecessary as new versions which work with the format are released.

Static Panel Image (.snap URL)

The `.snap` URL signals the server to return a static image of the front panel of a VI currently in memory. The query parameters in the URL specify the VI name and the attributes of the image.

You must open the front panel of the VI to take snapshots for static images because closed panels do not update the images of controls when their value changes.

Syntax

```
.snap?VI_Name
    [&type=type]
    [&depth=depth]
    [&quality=quality]
    [&compression=compression]
```

`VI_Name`—The name of the returned VI panel. You must encode the VI name according to HTTP conventions, that is replace special characters with `%xx`, where `xx` is the character's hexadecimal value. You can use the CGI Escape HTTP Param VI to encode the VI name.

`type`—The returned image type, either JPEG or PNG. If no `type` is specified, the default `type` is used. Refer to the *PanelImageType* directive in Appendix A, *Configuration Directives*, for more information.

`depth`—The depth of the returned image. `depth` can be 1, 4, 8, or 24 bits. If no `depth` is specified, the default `depth` is used. Refer to the *PanelImageDepth* directive in Appendix A, *Configuration Directives*, for more information.

`quality`—The image quality and memory size of the panel image. `quality` can be between 0 and 100. If no `quality` is specified, the default `quality` is used. Refer to the *PanelImageQuality* directive in Appendix A, *Configuration Directives*, for more information.

`compression`—The compression level used for compressing PNG images. `compression` can be between 0 and 7. If no compression is specified, the default PNG `compression` is used.

Examples

- To return the panel image of the VI `My VI.vi` from the machine `foo` using the default image `type`, `depth` and `quality`, use the following code:

  ```
  http://foo/.snap?My%20VI.vi
  ```

- To return the panel image of the VI `Test 1.vi` from the machine `foo` using image `depth = 24` and image `type = PNG`, use the following code:

  ```
  http://foo/.snap?Test%201.vi&depth=24&type=png
  ```

- To embed the image of the VI `Example.vi` running on the same machine as the page containing this HTML tag, use the following code:

  ```
  <IMG SRC="/.snap?Example.vi">
  ```

- To embed the image of the VI `Example.vi` running on the machine `foo`, use the following code:

  ```
  <IMG SRC="http://foo/.snap?Example.vi">
  ```

Animated Panel Image (.monitor URL)

The `.monitor` URL signals the server to return an animated image of the front panel of a VI currently in memory. The query parameters in the URL specify the VI name, attributes of the animation, and attributes

of the image. The server accomplishes this animation by taking subsequent snapshots of the panel image and sending them to the client. Only clients that work with server-push animations, such as Netscape Navigator, receive the animated image; other browsers receive one static image.

You must open the front panel of the VI to take snapshots for animated images because closed panels do not update the images of controls when their value changes.

Syntax

```
.monitor?VI_Name
    [&refresh=refresh]
    [&lifespan=lifespan]
    [&type=type]
    [&depth=depth]
    [&quality=quality]
    [&compression=compression]
```

`VI_Name`—The name of the returned VI panel. You must encode the VI name according to HTTP conventions; that is, replace special characters with `%xx` where `xx` is the hexadecimal value of the character. You can use the VI `CGI Escape HTTP Param.vi` to encode the VI name.

`refresh`—The number of seconds after which a new snapshot is sent. If no `refresh` is specified, the default `refresh` rate is used. Refer to the *ServerPushRefresh* directive in Appendix A, *Configuration Directives*, for more information.

`lifespan`—The number of seconds the panel animation lasts. `lifespan=0` implies that the animation continues until the browser cancels it. If no `lifespan` is specified, the default `lifespan` is used. Refer to the *ServerPushLifespan* directive in Appendix A, *Configuration Directives*, for more information.

`type`—The returned image type, either JPEG or PNG. If no `type` is specified, the default `type` is used. Refer to the *PanelImageType* directive in Appendix A, *Configuration Directives*, for more information.

`depth`—The depth of the returned image. `depth` can be 1, 4, 8, or 24 bits. If no `depth` is specified, the default `depth` is used. Refer to the

PanelImageDepth directive in Appendix A, *Configuration Directives*, for more information.

`quality`—The image quality and memory size of the JPEG panel image. `quality` can be between 0 and 100. If no `quality` is specified, the default `quality` is used. Refer to the *PanelImageQuality* directive in Appendix A, *Configuration Directives*, for more information.

`compression`—The compression level used for compressing PNG images. `compression` can be between 0 and 7 bits. If no compression is specified, the default PNG `compression` is used.

Examples

- To generate an animated panel image of the VI `My VI.vi` from the machine foo, using the default `refresh`, `lifespan`, image `type`, `depth`, and `quality`, use the following code:

  ```
  http://foo/.monitor?My%20VI.vi
  ```

- To generate a 60-second animation of the panel image of the VI `Test 1.vi` from the machine `foo`, using the default image `type` and `quality`, but using `refresh=5`, use the following code:

  ```
  http://foo/.smonitor?Test%201.vi
  &refresh=5&lifespan=60
  ```

- To embed an animation of the VI `Example.vi` running on the same machine as the page containing this HTML tag, use the following code:

  ```
  <IMG SRC="/.monitor?Example.vi">
  ```

- To embed an animation of the VI `Example.vi` running on the machine `foo`, use the following code:

  ```
  <IMG SRC="http://foo/.monitor?Example.vi
  &refresh=10">
  ```

Animated Panel Image (.spool URL)

The `.spool` URL signals the server to return animated panel images produced by a CGI VI. The query parameters in the URL specify the CGI, attributes of the animation, and the CGI parameters. The server accomplishes this animation by repeatedly calling the CGI and returning its output as a server-push animation. Only clients that work with server-push animations, such as Netscape Navigator, receive the animated image; other browsers receive one static image.

Syntax

```
.spool?CGI_path
    [&refresh=refresh]
    [&lifespan=lifespan]
    [&params]
```

`CGI_path`—The path to the CGI. You can make the path relative to the location of the document that embeds the `.spool` image tag.

`refresh`—The number of seconds after which a new snapshot is sent. If no `refresh` is specified, the default `refresh` rate is used. Refer to the *ServerPushRefresh* directive in Appendix A, *Configuration Directives*, for more information.

`lifespan`—The number of seconds the panel animation lasts. `lifespan=0` implies that the animation continues until you cancel it. If no `lifespan` is specified, the default `lifespan` is used. Refer to the *ServerPushLifespan* directive in Appendix A, *Configuration Directives*, for more information.

`params`—A list of parameters sent to the CGI, separated by an ampersand (`&`). If your CGI returns images of a VI panel, you can specify that the same image parameter as in the `.snap` URL be interpreted by the VI `CGI Get Panel Image.vi`.

Examples

- To generate an animation of the documents returned by the CGI VI `/cgi-bin/movie.vi` using the default `refresh` and `lifespan`, use the following code:

  ```
  http://foo/.spool?/cgi-bin/movie.vi
  ```

- To generate an animation of the documents returned by the CGI VI `/cgi-bin/data.vi` using the default `refresh` and `lifespan`, use the following code:

  ```
  http://foo/.spool?/cgi-bin/data.vi
  &base=2.3&offset=1000
  ```

  The CGI receives the parameters `base` and `offset` with the values `2.3` and `1000`, respectively.

- To generate a 60 second animation of the documents returned by the CGI VI `/cgi-bin/movie.vi` from the machine `foo`, use the following code:

  ```
  http://foo/.spool?/cgi-bin/movie.vi&refresh=5
  &lifespan=60&name=John%20Lennon
  ```

  The CGI receives the parameter **name** with the value `John Lennon` and `refresh=5`.

- To embed an animation of the documents returned by the CGI VI `test.vi`, located in the same directory as the page containing this HTML tag, use the following code:

  ```
  <IMG SRC="/.spool?test.vi">
  ```

- To embed an animation of the documents returned by the CGI VI `/cgi-bin/movie.vi` from the same machine as the page containing this HTML tag, use the following code:

  ```
  <IMG SRC="/.spool?/cgi-bin/movie.vi">
  ```

- To embed an animation of the documents returned by the CGI VI `/cgi-bin/movie.vi` from the machine `foo`, use the following code:

  ```
  <IMG SRC="http://foo/.spool?/cgi-bin/movie.vi">
  ```

What Is the Common Gateway Interface?

The Common Gateway Interface (CGI) is a standard for external gateway programs to interface with information servers such as HTTP servers. The current version is CGI/1.1. You can find additional documentation and specifications for the CGI on the National Center for Supercomputing Applications (NCSA) HTTPd web site, at the following address:

```
<http://hoohoo.ncsa.uiuc.edu/cgi>
```

☞ **Note:** ***The term CGI often is used to describe a CGI application as well as the interface to that application.***

On the World Wide Web, when a client sends a request whose URL specifies a CGI application, the server decodes the request, loads the application, and then executes the application. The application, in turn, generates data and returns it to the server. The server then sends a reply containing this data to the client who displays it. Figure 6-3, *CGI Application Process*, shows how this process works.

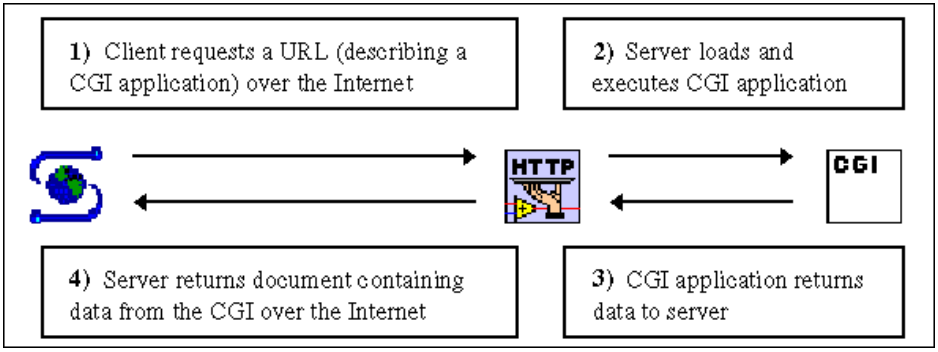

Figure 6-3. CGI Application Process

You can generate documents dynamically using CGI applications. This process can help you when the data in your documents changes over time or when you generate the document according to user-supplied criteria.

You can supply parameters for your CGI applications in the following two ways, depending on the method with which a CGI is invoked.

- `POST` method—Used only with HTML forms, when the user clicks on a **Submit** button.
- `GET` method—Used mainly in HTML links, but also in HTML forms.

When the user fills out your HTML `POST` form and clicks on the **Submit** button, the browser encodes the contents of the fields into an ampersand (`&`) separated list of `name=value` parameter pairs and sends this string as the content of the `POST` request. If your form contains the fields `name` and `age` and the user enters `John Smith` and `27`, respectively, the browser sends the string `name=John%20Smith&age=20`. The string `%20` in the name represents a space because `20` is the hexadecimal ASCII value of the space character.

With the `GET` method, you use a question mark to separate the CGI parameters from its name. When your form uses the `GET` method to invoke a CGI application, the encoded string is appended to the name of the CGI. Most often, though, an HTML link with the parameters explicitly specified invokes a CGI through the `GET` method. For example, the following URL connects to the server `some.server.adr`,

invokes the CGI `/cgi-bin/example.vi`, and passes the parameter `single parameter`.

```
<http://some.server.adr/cgi-bin/
example.vi?single%20parameter>
```

Similarly, the following URL invokes the CGI with two parameters **name** and **age** with value `John Smith` and `27`, respectively.

```
<http://some.server.adr/cgi-bin/
example.vi?name=John%20Smith&age=20>
```

CGI VIs With the HTTP Server

According to the CGI specification, a server and a CGI application communicate through environmental variables and through standard inputs and outputs. Specifically, when the server executes a CGI application, it passes it information in the environmental variables, shown in Table 6-1. The CGI application also can read data from standard input and write any data it generates to standard output.

Table 6-1. CGI Environment Variables

Name	Description
GATEWAY_INTERFACE	Version of the interface, currently `CGI/1.1`.
SERVER_SOFTWARE	Name and version of the G Web Server.
SERVER_NAME	Name of the machine running the G Web Server as configured or determined by the server.
SERVER_PORT	TCP port at which the server listens for requests.
DOCUMENT_ROOT	Root directory (in UNIX format) of your server documents.
REMOTE_HOST	Domain name or IP address of the remote machine that the client uses to connect.
REMOTE_ADDR	IP address of the remote machine that the client connects from.
SCRIPT_NAME	Virtual path to the CGI VI.
REQUEST_METHOD	Method by which you invoke the CGI, either `GET` or `POST`.
SERVER_PROTOCOL	Protocol over which the client communicates with the server, currently `HTTP/1.0`.
HTTP_REFERER	URL of document which contains the link that invoked this CGI.
HTTP_USER_AGENT	Browser software used by the remote client.
HTTP_ACCEPT	List of MIME-like types that the browser understands.
QUERY_STRING	URL-encoded parameters sent to this CGI.

Table 6-1. CGI Environment Variables (Continued)

Name	Description
REMOTE_USER	User name of client.
REMOTE_IDENT	User password of client.

Because the G environment does not work with standard input and output, a CGI VI receives the standard input data as a string when it receives a request. Similarly, instead of writing to standard output, it passes any data it generates as a string.

Figure 6-4 shows the basic structure of a CGI application.

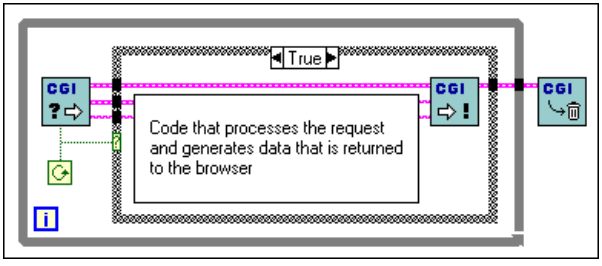

Figure 6-4. Structure of a CGI Application

This CGI application consists of the three VIs listed below. The VIs inside the loop receive requests from the G Web Server and return replies. The VI outside the loop informs the G Web Server that this CGI application finished executing and that it can be released from memory. You can find these VIs in the **User Libraries»Internet Toolkit»CGI VIs»CGI Template VIs** palette. To access more detailed information about these VIs, right-click on the VI and select **Online Help** from the VI pop-up menu.

- CGI Read Request VI—Waits for and reads an HTTP request for the CGI VI.
- CGI Write Reply VI—Sends the HTTP reply to the browser.
- CGI Release VI—Informs the server that the CGI application finished executing.

What Are CGI Utility VIs?

Many CGI applications execute common tasks, such as looking at the `QUERY_STRING` environmental variable to perform their task. Because of this, the G Web Server contains VIs that implement these common functions. These CGI Utility VIs are grouped in the following areas:

- CGI Parameter VIs—Convert to and from URL-encoded parameters.
- File Path VIs—Convert between platform-specific and virtual or UNIX-format paths.
- Network VIs—Provide network support.
- Panel Image VIs—Produce images from other VI front panels.
- Keyed Array VIs—Build and index data in Keyed Array format.
- HTML VIs—Generate HTML code.
- State Information VIs—Maintain information between multiple CGI invocations.

☞ **Note:** ***For more information on the State Information VIs, refer to the section*** *How Do I Maintain Client State Information?* ***later in this chapter.***

CGI Parameter VIs

URLs do not permit many characters, such as spaces or ASCII characters with values greater than 127, as part of their code. Other characters, such as `?`, `&`, `/`, `:`, `;`, and `=` have special meaning when used with HTTP. Whenever you use these characters as part of a URL, you must encode them. You can use the CGI Parameter VIs to complete this encoding for you. You replace each special character with a percent (`%`) character followed by two hexadecimal digits describing its numeric value. You also can replace the space character with the plus (`+`) character. For example, in order to pass the string `War & Peace` to the CGI VI `test.vi`, the URL contains the following code:

```
test.vi?War%20%26%20Peace
```

When you pass several parameters to a CGI VI, you pass them as an ampersand (`&`) separated list of `name=value` pairs. For example, to pass the parameters **name** and **age** with values `John Smith` and `27` to the CGI VI `test.vi`, the URL contains the following code:

```
test.vi?name=John%20Smith&age=27
```

You can use the following VIs to convert between URL-encoded strings and normal data. You can find these VIs in the **User Libraries»Internet Toolkit»CGI VIs** palette. To access more detailed information about these VIs, right-click on the VI and select **Online Help** from the VI pop-up menu.

- CGI Escape URL Parameter VI—Encodes a string according to URL conventions.
- CGI Unescape URL Parameter VI—Decodes a URL-encoded string.
- CGI Parse URL-Encoded Param String VI—Converts a URL-encoded parameter string into a Keyed Array.
- CGI Build URL-Encoded Param String VI—Converts a Keyed Array into a URL-encoded parameter string.
- CGI Get Query Parameters VI—Converts the `QUERY_STRING` environmental variable into a plain-text string and parameter list.

File Path VIs

URLs consist of a method part, host, and a virtual path to a document, in UNIX format. For example, the URL `<http://www.natinst.com/labview/internet>` uses the `http` method, the host `www.natinst.com`, and the virtual path `/labview/internet`. A *virtual path* is a path the server can use to redirect the request to another URL, alias it into another path, or append it to the path of the server root directory. The File Path VIs convert your platform-specific paths to virtual and UNIX-style paths. For example, if you use Windows to run the server `www.natinst.com` with the root directory `c:/www`, the virtual path refers to the absolute path `c:/www/labview/internet`.

You can use the following VIs to convert between platform-specific paths and virtual and UNIX-style paths. You can find these VIs in the **User Libraries»Internet Toolkit»CGI VIs** palette. To access more detailed information about these VIs, right-click on the VI and select **Online Help** from the VI pop-up menu.

- CGI Get Info VI—Returns the CGI VI name, absolute path, and virtual path through which you invoked the application.
- CGI Unix to Path VI—Converts a string containing a UNIX-style path to a G path.
- CGI Path To Unix VI—Converts a G path to a UNIX-style path.

- CGI Build Unix Path VI—Builds a UNIX-style path from a base path and a second path. If the second path is relative, it is appended to the base path, if it is absolute, it is used instead of the base path.
- CGI Translate Virtual Path VI—Translates a virtual path to an absolute path.
- CGI Script Realive Path VI—Returns a path relative to the CGI application. This is useful when your CGI application stores data files in its own directory.

Network VIs

CGI applications can use the regular TCP/IP VIs to convert an IP address to a domain name or vice versa. However, the Network VI listed below might work better. This VI looks at the server **UseDNS** configuration directive setting and does not try to obtain a domain name when the Domain Name System (DNS) lookup is turned off. You can find this VI in the **User Libraries»Internet Toolkit»CGI VIs** palette. To access more detailed information about this VI, right-click on the VI and select **Online Help** from the VI pop-up menu.

- CGI IP Name To Name VI—Takes a domain name or IP address and returns its IP address and domain name. It only attempts DNS lookup according to the server configuration. Refer to the *UseDNS* directive in Appendix A, *Configuration Directives*, for more information.

Panel Image VIs

Using the Panel Image VIs, you can return the image of another VI front panel. With the Panel Image VIs, you can specify the name and certain image parameters to capture the VI front panel image. The VIs return the image data as well as an appropriate header string that correctly identifies the image type. You can use the following VIs to produce these front panel images. You can find these VIs in the **User Libraries»Internet Toolkit»CGI VIs** palette. To access more detailed information about these VIs, right-click on the VI and select **Online Help** from the VI pop-up menu.

- CGI Get Panel Image VI—Uses a VI name and a Keyed Array parameter and returns the image of the VI specified by the name with image parameters specified by the Keyed Array.

- CGI Get Panel JPEG Image VI—Uses a VI name and image parameters and returns the front panel image in JPEG format.
- CGI Get Panel PNG Image VI—Uses a VI name and image parameters and returns the front panel image in PNG format.

Keyed Array VIs

The G Web Server supplies the CGI environmental variables, listed in Table 6-1, *CGI Environment Variables*, in the form of a *Keyed Array*. A Keyed Array is a data structure similar to an array of strings. However, unlike a regular array, you index elements by a string key instead of numeric index. Keyed Arrays are similar to the *Associative Array* in the Perl language. Many operations on Keyed Arrays allow you to specify whether string matching should be case sensitive, the values for this parameter are `a != A` (case sensitive), `a == A` (case insensitive), and `system` (same case sensitive as when comparing file names, for example, case sensitive on UNIX and case insensitive on Windows and MacOS).

You can use the following VIs to index, add, or remove elements and access the keys and values stored in a Keyed Array. You can find these VIs in the **User Libraries»Internet Toolkit»CGI VIs»Keyed Array VIs** palette. To access more detailed information about these VIs, right-click on the VI and select **Online Help** from the VI pop-up menu.

- Keyed Array Add VI—Inserts a new element into a Keyed Array.
- Keyed Array Index VI—Indexes an element in a Keyed Array.
- Keyed Array Remove VI—Removes an element from a Keyed Array.
- Keyed Array Clear VI—Removes all elements from a Keyed Array.
- Keyed Array Keys VI—Returns an array of keys of all the elements in the Keyed Array.
- Keyed Array Values VI—Returns an array of values of all the elements in the Keyed Array.
- Keyed Array Contents VI—Returns an array of keys and an array of the corresponding values of all the elements in the Keyed Array.
- Keyed Array Index Wildcards VI—Indexes the keys of the elements in the Keyed Array wildcard expressions.
- Keyed Array Wildcard Index VI—Indexes the elements of a Keyed Array using wildcard expressions.
- Keyed Array Equal VI—Returns the value TRUE if all the elements in two Keyed Arrays contain the corresponding keys and values.

HTML VIs

The HTML VI library in this toolkit contains VIs that generate all the tags specified in HTML 3.2, with the exception of the obsolete tags `XMP`, `LISTING` and `PLAINTEXT`. This toolkit also contains a VI for building arbitrary HTML tags if you want to generate tags not in the HTML 3.2 specification. The library also includes VIs for building tags with attributes specific to the G Web Server (for panel images) and for converting arrays into HTML tables.

You wire the HTML VIs in sequence so each VI generates one or more HTML tags. Most VIs also have a **label** input so you can insert extra text between the incoming HTML code and the tag the VI generates. VIs that implement tags without a corresponding closing tag are wired in sequence. VIs that contain a closing tag and enclose some text or HTML code take the enclosed content as a parameter. Optional and required tag attributes are also specified as parameters. Some VIs generate more complex attribute strings for certain HTML tags. Text attributes, with the exception of URLs, are filtered so that the HTML reserved characters are encoded properly.

Figure 6-5 constructs the HTML code for the following text:

`These are` **`words`** `with` ***`style`***

This string also includes a horizontal rule and an embedded picture `example.gif`, with the alternate description `Taste & Style`.

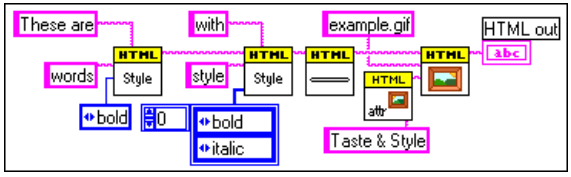

Figure 6-5. HTML Block Diagram

The following is the output of the **HTML out** string:

```
These are<B>words</B>with<I><B>style</B></I><HR>
<IMG SRC="example.gif" ALT="Taste &amp; Style">
```

Some tags only make sense within the content of some other tag. For example, an HTML table is defined by the `<TABLE>` tag which encloses a list of table row tags `<TR>` which in turn enclose table cell tags `<TD>`. Figure 6-6 constructs the HTML code for the following table.

One	Two
1	2

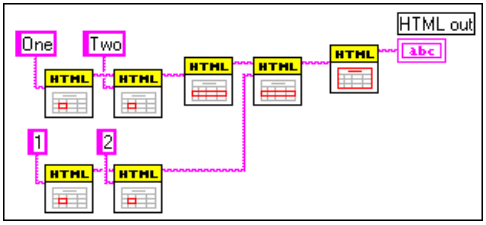

Figure 6-6. HTML Table Block Diagram

The following is the output of the **HTML out** string:

```
<TABLE>
<TR>
<TD>One</TD>
<TD>Two</TD>
</TR>
<TR>
<TD>1</TD>
<TD>2</TD>
</TR>
</TABLE>
```

Because some HTML code, such as building tables out of arrays, is fairly common, this toolkit includes VIs that combine several HTML tags to construct more complicated HTML code. For example, the VI `HTML+ String Array To Table.vi` constructs a table out of a two dimensional string array. Figure 6-7, *HTML+ String Array to Table VI*, generates the same HTML code shown in Figure 6-6.

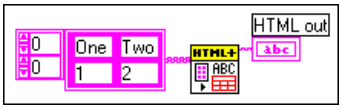

Figure 6-7. HTML+ String Array to Table VI

Some VIs generate tags with parameters or attributes specific to the G Web Server. For example, the `HTML+ Monitor.vi` generates an image tag for a VI front panel animated image. Figure 6-8, shown below, constructs HTML code for an animation of the VI `animation example.vi` that refreshes every 5 seconds.

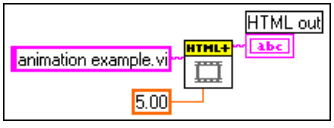

Figure 6-8. HTML+ Monitor VI

The following is the output of the **HTML out** string:

```
<IMG SRC="/.monitor?animation%20example.vi
&refresh=5.000">
```

Using `HTML Generic Tag.vi`, you can create any HTML tag. You also can use many of the CGI Utility VIs to help you construct tag attributes. In Figure 6-9, shown below, the diagram on the left uses the VI `HTML Font.vi` to generate a tag for colored text. The diagram on the right constructs the same HTML code using the VIs `HTML Color Tag Attribute.vi` and `HTML Generic Tag.vi`.

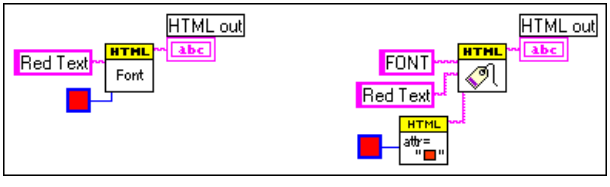

Figure 6-9. HTML Generic Tag VI

The following is the output of the **HTML out** string:

```
<FONT COLOR="#FF0000">Red Text</FONT>
```

The VI `HTML Document.vi` uses the HTML code you created and builds an HTML document out of it. For example, Figure 6-10, shown below, builds an HTML document with the title `Acquired Data`, a heading also called `Acquired Data`, a table built from a two dimensional numeric array with 3 digits of precision, and a link to the National Instruments web site called `National Instruments`.

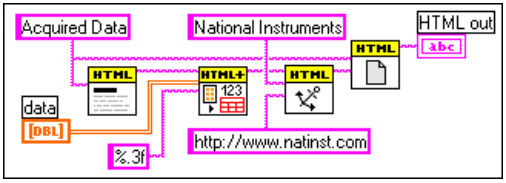

Figure 6-10. HTML Document VI

The following is the output of the **HTML out** string:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2
Draft//EN">
<HTML>
<!-- Constructed with the G Web Server -->
<HEAD>
<TITLE>Acquired Data</TITLE>
</HEAD>

<BODY >
<H1>Acquired Data</H1>
<TABLE>
<TR>
<TD>1.000</TD>
<TD>2.000</TD>
</TR>
<TR>
<TD>3.000</TD>
<TD>4.000</TD>
</TR>
</TABLE>
<A HREF="http://www.natinst.com">National
Instruments</A>
</BODY>
</HTML>
```

You can use the following VIs to construct your HTML code. You can find these VIs in the **User Libraries»Internet Toolkit»CGI VIs»HTML VIs** palette. To access more detailed information about these VIs, right-click on the VI and select **Online Help** from the VI pop-up menu.

- HTML Address VI—Constructs the `<ADDRESS>` tag.
- HTML Alignment Tag Attribute VI—Constructs the horizontal `ALIGN` tag attribute.
- HTML Anchor VI—Constructs the `<A>` tag.
- HTML Applet VI—Constructs the `<APPLET>` tag.
- HTML Area VI—Constructs the image map `<AREA>` tag.
- HTML Base VI—Constructs the `<BASE>` tag for the HTML `HEAD` section.
- HTML Basefont VI—Constructs the `<BASEFONT>` tag.
- HTML Blockquote Text VI—Constructs the `<BLOCKQUOTE>` tag.
- HTML Break VI—Constructs the `<BR>` tag.
- HTML Build HREF Parameters VI—Constructs a CGI URL out of a base URL, a query string, and a parameter list.
- HTML Color Tag Attribute VI—Constructs a `COLOR` tag attribute from a numeric color value.
- HTML Definition List VI—Constructs a `<DL>` tag with embedded `<DT>` and `<DD>` tags.
- HTML Division VI—Constructs a `<DIV>` tag.
- HTML Document VI—Constructs an HTML documents out of HTML code.
- HTML Filter Special Characters VI—Replaces `&`, `"`, `<`, and `>` with `&`, `"`, `<`, and `>`, respectively.
- HTML Font Style VI—Constructs `<TT>`, `<I>`, `<B>`, `<U>`, `<STRIKE>`, `<BIG>`, `<SMALL>`, `<SUB>`, or `<SUP>` tags.
- HTML Font VI—Constructs a `<FONT>` tag.
- HTML Form Button VI—Constructs an HTML form `<INPUT>` tag of type `SUBMIT` or `RESET`.
- HTML Form Control Button VI—Constructs an HTML form `<INPUT>` tag of type `CHECKBOX` or `RADIO`.
- HTML Form File VI—Constructs an HTML form `<INPUT>` tag of type `FILE`.

- HTML Form Hidden Field VI—Constructs an HTML form <INPUT> tag of type HIDDEN.
- HTML Form Image Button VI—Constructs an HTML form <INPUT> tag of type IMAGE.
- HTML Form Radio Button Group VI—Constructs a list of HTML form <INPUT> tags of type IMAGE.
- HTML Form Selection VI—Constructs an HTML form <SELECT> tag with a list of embedded <OPTION> tags.
- HTML Form Text Area VI—Constructs an HTML form <TEXTAREA> tag.
- HTML Form Text VI—Constructs an HTML form <INPUT> tag of type TEXT or PASSWORD.
- HTML Form VI—Constructs an HTML <FORM> tag.
- HTML Generic Tag VI—Constructs a generic HTML tag.
- HTML Heading VI—Constructs an <H1>, <H2>, <H3>, <H4>, <H5>, or <H6> tag.
- HTML Horizontal Rule VI—Constructs an <HR> tag.
- HTML Image Attributes VI—Constructs ISMAP, USEMAP, ALT, ALIGN, BORDER, HSPACE, VSPACE, HEIGHT, and WIDTH image tag attributes.
- HTML Image VI—Constructs an <IMAGE> tag.
- HTML IsIndex VI—Constructs an <ISINDEX> head tag.
- HTML Link VI—Constructs a <LINK> head tag.
- HTML List VI—Constructs an <UL>, <OL>, <DIR>, or <MENU> tag with embedded <LI> tags.
- HTML Map VI—Constructs an image <MAP> tag.
- HTML Meta VI—Constructs a <META> head tag.
- HTML Numeric Tag Attribute VI—Constructs a numeric tag attribute.
- HTML Numeric Tag Attributes VI—Constructs a list of numeric tag attributes.
- HTML Paragraph VI—Constructs a <P> tag.
- HTML Param VI—Constructs an applet <PARAM> tag.
- HTML Phrase Style VI—Constructs an <EM>, <STRONG>, <DFN>, <CODE>, <SAMP>, <KBD>, <VAR>, or <CITE> tag.
- HTML Preformatted Text VI—Constructs a <PRE> tag.

- HTML Table Cell VI—Constructs a table `<TD>` or `<TH>` tag.
- HTML Table Row VI—Constructs a table `<TR>` tag.
- HTML Table VI—Constructs a `<TABLE>` tag.
- HTML Text Tag Attribute VI—Constructs a text tag attribute.
- HTML Text Tag Attributes VI—Constructs a list of text tag attributes.
- HTML+ Form Hidden Field List VI—Constructs a list of HTML form `<INPUT>` tags of type `HIDDEN`
- HTML+ Keyed Array To Table VI—Constructs a `<TABLE>` tag with embedded `<TR>` and `<TD>` tags from a Keyed Array.
- HTML+ Labeled Table VI—Constructs a `<TABLE>` tag with embedded `<TR>`, `<TH>`, and `<TH>` tags from a 2D array of strings, and arrays of strings describing the row and column headers.
- HTML+ Monitor VI—Constructs an `<IMG>` tag with a `.monitor` URL.
- HTML+ Numeric Array To Table VI—Constructs a `<TABLE>` tag with embedded `<TR>` and `<TD>` tags from a 2D array of doubles.
- HTML+ Param List VI—Constructs a CGI URL from a base URL, a query string, and a Keyed Array.
- HTML+ Snapshot VI—Constructs an `<IMG>` tag with a `.snap` URL.
- HTML+ Spool VI—Constructs an `<IMG>` tag with a `.spool` URL.
- HTML+ String Array To Table VI—Constructs a `<TABLE>` tag with embedded `<TR>` and `<TD>` tags from a 2D array of strings.

How Do I Maintain Client State Information?

The HTTP is a stateless protocol. Each time a client wants a document from a server, the client must establish a new connection and send a request. The server then receives the request, returns a reply, and closes the connection. The server does not maintain state information between individual connections.

Often, it is useful to maintain state information across several connections. For example, you are an online vendor who uses a "shopping cart" for your users. When a user browses through your online catalog, the user can add items he/she wants to purchase to a shopping cart. At the end, the user can choose to purchase the items in

the cart. You must maintain the information about the items until the user finishes the purchase. You can store information such as what items the user chose in a simple data file. However, this does not work if more than one user is shopping at a time. Because you want to work with multiple users, you must find an alternative information storage method.

You can choose from several approaches to maintain client state information across multiple accesses. For example, you can insert information you collect into hidden fields of an HTML form. Or, you could use a *cookie*. A cookie is a token that uniquely identifies some information. In the shopping cart example, the cookie records all the items the user has put into the shopping cart. Using a cookie, you can maintain your information on the client side or server side.

Client-Side Cookies

To store state information on the client machine, you can use client-side cookies. However, not all browsers work with client-side cookies. Some users do not want information written to their disk without their knowledge, and potential security risks can arise when client state information can be sent to a host other than the one storing the information.

Server-Side Cookies

To store state information on the server machine, you can use server-side cookies. There are several advantages to using server-side cookies:

- No dependence on the browser software
- No data stored on the client machine
- Less transmitted data
- Security maintained by the server

However, the disadvantages is that server state information is only maintained for a certain time, after which it automatically expires. This time is specified by the server and not the client.

Using Cookies

Refer to the previously mentioned online catalog example. The user enters your online store by clicking on a link to the main document. A CGI generates the main document by creating a cookie and embedding it in the document. Then, each time a user moves to another page on the server, the cookie is passed along with the request for that page. When the user adds an item to their shopping cart, the CGI handling that request adds the item to the information maintained by this cookie. When the user is ready to purchase the items, the CGI generating the purchase form reviews all the items associated with the cookie and builds the appropriate invoice.

Cookie Example

The first page of your online store might have a form on which users fill in their names and then click on a button. A CGI application processes this form and returns a document (containing the user name in appropriate places) in which all links are tagged with the cookie the CGI application generated, as shown below:

```
<A HREF="page.vi?cookie=02A34CCD&name=strings>
String Instruments</A><BR>
<A HREF="page.vi?cookie=02A34CCD&name=brasss>Brass
Instruments</A><BR>
<A HREF="page.vi?cookie=02A34CCD&name=woods>
Woodwind Instruments</A><BR>
```

The VI `CGI page.vi` looks at its parameters to generate a new page in which it embeds the cookie. If another user connects to your store, that person receives the same pages except with a different cookie value.

Cookie VIs

The G Web Server provides VIs for cookie management. With these VIs, you can create and destroy cookies, as well as add and query information associated with cookies. Several VIs help you create HTTP-connection based cookies and documents containing cookies.

The G Web Server defines a cookie as a cluster of two strings, *cookie ID* and *address*. The cookie ID string is used to identify a specific cookie and usually is embedded in the HTML document a CGI application creates. The address string ensures that only clients with the same address can access a cookie. This way, cookies created by requests from a specific client can be viewed and modified by that client only.

The following VIs create, modify and dispose of cookies. You can find these VIs in the **User Libraries»Internet Toolkit»CGI VIs»Cookie VIs** palette. To access more detailed information about these VIs, right-click on the VI and select **Online Help** from the VI pop-up menu.

- Cookie Create VI—Creates a unique cookie identifier to an empty cookie.
- Cookie Destroy VI—Disposes of a cookie.
- Cookie Add Entry VI—Adds an entry specified by a key and a value to a cookie.
- Cookie Get Entry VI—Returns the value of the entry specified by a key from a cookie.
- Cookie Get Data VI—Returns a Keyed Array containing all the entries in a cookie.
- Not A Cookie VI—Returns an invalid cookie.
- Not A Cookie? VI—Tests the validity of a cookie.

The G Web Server also provides several CGI Utility VIs that work with cookies in CGI applications. These VIs identify the cookies as parameters with the name `Magic_Cookie`.

You can use the following VIs to work with cookies in HTML documents. You can find these VIs in the **User Libraries»Internet Toolkit»CGI VIs** palette. To access more detailed information about these VIs, right-click on the VI and select **Online Help** from the VI pop-up menu.

- CGI Cookie VI—Returns the cookie associated with the CGI request, or creates a new cookie, given the **env** parameter and parameters of a CGI request, if there is a parameter `Magic_Cookie.`
- CGI Spool Cookie VI—Returns or creates a cookie associated with a `.spool` CGI request, based on the **cgi connection info** parameter returned from the VI `CGI Read Request.vi`.
- CGI Add Params To Cookie VI—Adds parameters to a cookie.
- CGI Build Cookie Document VI—Loads a file from disk and replaces every occurrence of the string `-*Cookie*-` with a cookie ID.

Configuration Directives

Appendix A

This appendix describes the configuration directives for the G Web Server and the use of wildcard expressions within the directives.

Wildcard Expressions

The G Web Server works with wildcard expressions you can use to specify patterns that match strings. The G Web Server wildcard expressions use formats similar to UNIX shell wildcard patterns. Both use the characters `*` and `?`, where `*` specifies 0 or more of any character and `?` specifies exactly one instance of any character. To match one of the wildcard characters, begin the expression with a `\`.

The following list provides some examples of wildcard expressions:

- `foo*` matches `foo`, `fooa`, and `foobar`, but does not match `afoo`.
- `*/.??*` matches any substring which contains `/.` followed by two or more characters. You can use this pattern for matching UNIX hidden files such as `/foo/bar/.htaccess`.
- `*\?` matches any substring ending in a question mark.

You can use these wildcard expressions in many of the following configuration directives.

Access Configuration Directives

The Access Configuration directives control the following aspects of document directories in a tree with the server:

- Access—Restricts access to a branch of the directory tree to allowed hosts and/or authenticated users.
- Server Features—Secures directories by disabling certain server functions.

For more information on the Access Configuration specification, refer to the section *Access Configuration*, in Chapter 6, *G Web Server*.

Directory

The **Directory** sectioning directive specifies the directory to which access control directives apply.

Scope

This directive only applies to the Global Access Configuration file (ACF). You must store all directives in the Global ACF in a **Directory** or **Panel** section.

Syntax

Opening Directive:

```
<Directory dir>
```

`dir` is the absolute path name of the directory you are protecting. It also can be a wildcard expression of a set of directories you want to protect.

Closing Directive:

```
</Directory>
```

File Name

`access.cfg`

Example

The following directive specifies that the directives contained in the **Directory** section only apply to the server directory `/c/web` and its subdirectories:

```
<Directory /c/web>
allow from *.natinst.com
</Directory>
```

Panel

The **Panel** sectioning directive specifies the VI panel to which access control directives apply.

Scope

This directive only applies to the Global ACF. You must store all directives in the Global ACF in a **Directory** or **Panel** section.

Syntax

Opening Directive:

```
<Panel vi>
```

`vi` is the name of the VI you are protecting. It also can be a wildcard expression of a set of VIs you want to protect.

Closing Directive:

```
</Panel>
```

File Name

`access.cfg`

Example

The following directive specifies that the directives contained in the **Panel** section only apply to those VIs ending with `.vi`:

```
<Panel *.vi>
deny from all
</Directory>
```

AllowOverride

The **AllowOverride** directive controls which Access Configuration directives you can overrule on a per-directory ACF.

You cannot restrict the Global ACF with this directive.

Scope

This directive only appears in the Global ACF.

Syntax

```
AllowOverride or1 or2 ... orn
```

`or` is one of the following:

- `None`—You cannot use ACFs in this directory.
- `All`—You can use any ACF in this directory.
- `AuthConfig`–You can use the following directives:
  - **AuthName**
  - **AuthType**

  - **AuthUserFile**
  - **AuthGroupFile**
- `Limit`—You can use the **Limit** sectioning directive.

File Name

`access.cfg`

Default

If no **AllowOverride** is specified, the server assumes the following:

```
AllowOverride All
```

Example

The following directive overrides the **Limit** sectioning directive for the ACFs in this directory:

```
AllowOverride Limit
```

AuthName

The **AuthName** directive sets the name of the authorization realm for this directory or panel. This realm is a name given to users so they know which username and password to send.

Scope

This directive applies to both Global and per-directory ACFs.

The **AuthType**, **AuthUserFile**, and **AuthGroupFile** directives must accompany this directive for user authentication to work properly.

Syntax

```
AuthName name
```

`name` is a short name describing this authorization realm. `name` might contain spaces.

File Name

`access.cfg`

Default

There is no default.

Example

The following directive sets the authorization name of this directory or panel to `Research Group`:

```
AuthName Research Group
```

AuthType

The **AuthType** directive sets the type of authorization used in this directory or panel.

Scope

This directive applies to both Global and per-directory ACFs.

The **AuthName**, **AuthUserFile**, and **AuthGroupFile** directives must accompany this directive for user authentication to work properly.

Syntax

`AuthType type`

`type` is the authentication type to use for this directory or panel. Currently, `type` is set to `Basic`.

File Name

`access.cfg`

Default

There is no default.

Example

The following directive sets the authorization type of this directory to `Basic`:

```
AuthType Basic
```

AuthUserFile

The **AuthUserFile** directive specifies the file to use as a list of users and passwords for user authentication.

You can use the Password Editor VI to manage your password files.

Scope

This directive applies to both Global and per-directory ACFs.

The **AuthName**, **AuthType**, and **AuthGroupFile** directives must accompany this directive for user authentication to work properly.

Syntax

`AuthUserFile path`

`path` is the absolute path of a user file created with the HTTP Password Support VI.

File Name

`access.cfg`

Default

There is no default.

Example

The following directive sets the authorization user file for this directory or panel to `/c/labview/user.lib/internet/http/conf/passwd.txt`:

```
AuthUserFile /c/labview/user.lib/internet/http/conf/
passwd.txt
```

AuthGroupFile

The **AuthGroupFile** directive specifies the file to use as a list of user groups for user authentication.

Scope

This directive applies to both Global and per-directory ACFs.

The **AuthName**, **AuthType**, and **AuthUserFile** directives must accompany this directive for user authentication to work properly.

Syntax

`AuthGroupFile path`

`path` is the absolute path of a group file to use in this directory or panel.

File Name

`access.cfg`

Default

There is no default.

Example

The following directive sets the authorization group file for this directory or panel to `/c/labview/internet/http/conf/group.txt`:

```
AuthGroupFile/c/labview/internet/http/conf/group.txt
```

Limit

The **Limit** sectioning directive controls which clients can access a directory or panel.

Scope

This directive applies to both Global and per-directory ACFs.

The **AuthName**, **AuthType**, and **AuthUserFile** directives must accompany this directive for user authentication to work properly.

Syntax

Opening Directive:

```
<Limit meth1 meth2 ... methn>
```

Each `meth` is one of the following methods:

- `GET` (used in the **Directory** section)—Clients can retrieve documents and execute CGIs.
- `POST` (used in the **Directory** section)—Clients can use `POST` CGIs.
- `Snapshot` (used in the **Panel** section)—Clients can receive static images of VI panels.
- `Monitor` (used in the **Panel** section)—Clients can receive server-push animations of VI panels.

Closing Directive:

```
</Limit>
```

You can use only the following directives inside the **Limit** section:

- **Order**
- **Deny**
- **Allow**
- **Require**

- **Satisfy**
- **Referer**
- **OnDeny**

File Name

`access.cfg`

Example

The following directive specifies that the only clients who can use the `.snap` command on this panel must be from `natinst.com` and authenticated group developers:

```
<Limit Snapshot>
order deny,allow
deny from all
allow from .natinst.com
require group developers
</Limit>
```

Order

The **Order** directive determines the order in which the **Deny** and **Allow** directives are evaluated within a **Limit** section.

Scope

You can use this directive only within **Limit** sections.

Syntax

`order ord`

`ord` is one of the following:

- `deny,allow`—In this case, you first evaluate all the **Deny** directives and then all the **Allow** directives. This order makes the **Deny** directives the default, and the **Allow** directives the exceptions.
- `allow,deny`—In this case, you first evaluate all the **Allow** directives and then all the **Deny** directives. This order makes the **Allow** directives the default, and the **Deny** directives the exceptions.
- `mutual-failure`—In this case, you specify specific hosts to be allowed or denied. Any host appearing on the **Allow** list is allowed, and any host on the **Deny** list is denied. If a host does not appear on either list, it is denied.

File Name

`access.cfg`

Default

If no order is given, the server assumes the following:

`order deny,allow`

Example

The following directive specifies the order `deny,allow` for this directory. In this command, the server first evaluates the **Deny** directive for the `GET` method, which denies all access to this directory. Then, the server evaluates the **Allow** directive, which allows clients from `.natinst.com` to `GET` files from this directory.

```
<Limit GET>
order deny,allow
deny from all
allow from .natinst.com
</Limit>
```

Deny

The **Deny** directive determines which hosts can access a given directory with a given method.

Scope

You can use this directive only within **Limit** sections.

Syntax

`deny from host1 host2 ... hostn`

`host` is one of the following:

- A domain name—You can specify a domain name, such as `.natinst.com`. The host name must end in a domain name to be denied.
- A host name—You can specify a full host name, such as `eagle.natinst.com`.
- A partial IP address—You can specify the first 1–3 octets of an IP address, for subnet restriction.

☞ **Note:** ***Because comparison is done by string matching, you must use trailing dots to match octets. For example,*** `130.164.1` ***matches host*** `130.164.12.18`***, but*** `130.164.1.` ***does not.***

- The keyword `all`—You can specify the keyword `all` to deny all hosts.

File Name

`access.cfg`

Default

The default is to restrict to none.

Example

The following directive specifies the order `deny,allow` for this directory. In this command, the server first evaluates the **Deny** directive for the `GET` method, which denies all access to this directory. Then, the server evaluates the **Allow** directive, which allows clients from `.natinst.com` to `GET` files from this directory.

```
<Limit GET>
order deny,allow
deny from all
allow from .natinst.com
</Limit>
```

Allow

The **Allow** directive determines which hosts can access a given directory with a given method.

Scope

You can use this directive only within **Limit** sections.

Syntax

`allow from host1 host2 ... hostn`

`host` is one of the following:

- A domain name—You can specify a domain name, such as `.natinst.com`. The host name must end in a domain name to be denied.
- A host name—You can specify a full host name, such as `eagle.natinst.com`.
- A partial IP address—You can specify the first 1–3 octets of an IP address, for subnet restriction.

☞ **Note:** ***Because comparison is done by string matching, you must use trailing dots to match octets. For example,*** `130.164.1` ***matches host*** `130.164.12.18`***, but*** `130.164.1.` ***does not.***

- The keyword `all`—You can specify the keyword `all` to deny all hosts.

File Name

`access.cfg`

Default

The default is to allow from all.

Example

The following directive specifies the order `deny,allow` for this directory. In this command, the server first evaluates the **Deny** directive for the `GET` method, which denies all access to this directory. Then, the server evaluates the **Allow** directive, which allows clients from `.natinst.com` to `GET` files from this directory.

```
<Limit GET>
order deny,allow
deny from all
allow from .natinst.com
</Limit>
```

Require

The **Require** directive determines which authenticated users can access a given directory or panel with a given method.

Scope

You can use this directive only within **Limit** sections.

Syntax

`require entity en1 en2 ... enn`

`en` are entity names, separated by spaces.

`entity` is one of the following:

- `user`—Restricts access to this directory or panel, with the given methods, to only the named users.
- `group`—Restricts access to this directory or panel, with the given methods, to only the named group.
- `valid-user`—All the users defined in the **AuthUserFile** can access once providing a valid password.

File Name

`access.cfg`

Default

There is no default.

Example

The following directive specifies the order `deny,allow` and the **Require** directive for this directory. In this command, the server first evaluates the **Deny** directive for the `GET` method, which denies all access to this directory. Then, the server evaluates the **Allow** directive, which allows clients from `.natinst.com` to `GET` files from this directory. Finally, the server evaluates the **Require** user authentication directive and only allows users named `bob` or users in the group developers access to this directory.

```
<Limit GET>
order deny,allow
deny from all
allow from .natinst.com
require user bob
require group developers
</Limit>
```

Satisfy

The **Satisfy** directive determines the access to a directory if you use both the **Allow** and **Require** directives.

Scope

You can use this directive only within **Limit** sections.

Syntax

```
Satisfy all | any
```

`all` specifies that the user must satisfy both the **Allow** and **Require** directives to gain access to the directory.

`any` specifies that the user must only satisfy one of the specified **Allow** or **Require** directives.

File Name

`access.cfg`

Default

The default is `all`.

Example

The following directive restricts access for this directory to either users from `.natinst.com` or users named `bob`. In this command, the server first evaluates the **Deny** directive, which denies all access to this directory. Next, the server evaluates the **Allow** directive, which allows clients from `.natinst.com` to access files in this directory. Finally, the server evaluates the **Satisfy** directive, which allows access to any users from `.natinst.com`. If you do not meet the Satisfy directive, the server requires user authentication and only allows access to users named `bob`.

```
<Limit GET POST>
order deny,allow
deny from all
allow from .natinst.com
require user bob
satisfy any
</Limit>
```

OnDeny

The **OnDeny** directive provides non-HTTP based access control to a directory. This directive sends a browser which fails the **Limit** directive to a specified URL.

Scope

You can use this directive only within **Limit** sections.

Syntax

```
OnDeny URL
```

`URL` is the URL you redirect the browser to.

File Name

`access.cfg`

Default

The default is to let the normal HTTP 401 response handle the authentication failure.

Example

The following directive redirects any user connecting from outside of `natinst.com` to the URL `http://www.natinst.com/deny.htm`:

```
<Limit GET>
order deny,allow
deny from all
allow from .natinst.com
OnDeny http://www.natinst.com/deny.htm
</Limit>
```

Server Configuration Directives

The Server Configuration directives control the operation of the server. For more information on the Server Configuration specification, refer to the section *Server Configuration*, in Chapter 6, *G Web Server*.

Port

The **Port** directive sets what port the server listens on for clients.

Syntax

```
port num
```

`num` is a number from 0–65535. Most ports below 1024, are reserved by the system. For example, `Port 80` is reserved for HTTP.

To use a port below 1024 on UNIX and Windows NT, you might need to run the server as root or system administrator.

You can use only one **Port** directive in the configuration file.

Default

If you do not specify a **Port**, the server assumes the following:

```
Port 80
```

☞ **Note:** ***If you do not specify a* Port *other than the default, you must run the server as a root or system administrator on the UNIX or Windows NT platforms.***

Examples

- The following directive instructs the server to listen on port `8080`:

  ```
  Port 8080
  ```

- The following directive instructs the server to try to listen on port `84`:

  ```
  Port 84
  ```

UseDNS

The **UseDNS** directive specifies whether the server tries to get the DNS format for IP addresses when logging data. On machines not connected to a domain name server, turn this option OFF. It also can be turned OFF on machines that do have DNS access in order to improve performance, but this affects the access control based on DNS names.

Syntax

```
UseDNS setting
```

`setting` is either ON or OFF.

You can use only one **UseDNS** directive in the configuration file.

Default

If you do not specify to **UseDNS**, the server assumes the following:

```
UseDNS off
```

Example

The following directive instructs the server to use domain names when describing connections:

```
UseDNS on
```

ServerAdmin

The **ServerAdmin** directive provides your e-mail address to the server. The server uses this address to let people report error conditions.

Syntax

```
ServerAdmin address
```

`address` is an e-mail address.

Default

If you do not specify a **ServerAdmin** address, the server prints `no address` for reporting errors.

Example

The following directive prints errors with the address `webmaster@foo.com` as the contact person:

```
ServerAdmin webmaster@foo.com
```

ServerRoot

The **ServerRoot** directive specifies the directory location of the server files.

On start-up, the server expects to find the Server Configuration File as `conf/lvhttp.cfg` in the **ServerRoot** directory. Other Server Configuration directives might use this directory to provide relative paths for locations of files.

Syntax

```
ServerRoot dir
```

`dir` is an absolute path of a directory on your server machine.

You can use only one **ServerRoot** directive in the server configuration file.

Default

If you do not specify a **ServerRoot** directory, the server uses the directory in which the **ServerRoot** is located, as follows:

```
ServerRoot <default directory>/internet/http
```

Examples

- The following directive sets the **ServerRoot** to the directory `C:/toolkits/internet/http` on Windows:

  ```
  ServerRoot /c/toolkits/internet/http
  ```

- The following directive sets the **ServerRoot** to the directory `Macintosh HD:Web Stuff` on MacOS. You must escape the spaces in the path with a \.

  ```
  ServerRoot Macintosh\ HD/Web\ Stuff
  ```

ServerName

The **ServerName** directive sets the host name the server returns when creating redirection URLs. You use this directive on systems where the host name returned is a DNS alias, such as `www.natinst.com`.

Syntax

```
ServerName FQDN
```

`FQDN` is the full host name (including domain name) returned as the server address.

Default

If you do not specify a **ServerName**, the server attempts to retrieve it through the `IP To Name` function.

Example

The following directive sets the server host name as `www.natinst.com`:

```
ServerName www.natinst.com
```

☞ **Note:** ***The default ServerName for this host name is*** `webserver.natinst.com`***, as would be returned by the*** `IP To Name` ***function.***

StartServers

The **StartServers** directive determines how many connection handlers the server launches. More handlers require more memory but result in fewer pending requests.

Syntax

```
StartServers number
```

`number` is between 1 and 8, inclusive.

Default

If you do not specify a number, the server launches 4 handlers.

Example

The following directive launches 8 connection handlers at start-up:

```
StartServers 8
```

TimeOut

The **TimeOut** directive sets the amount of time the server waits for a client to send its query once the client connects to the server. **TimeOut** also can set the maximum amount of time the server spends waiting for a client to accept information.

Syntax

```
TimeOut time
```

`time` is the amount of time, in seconds, the server waits.

You can use only one **TimeOut** directive in the configuration file.

Default

If you do not specify a **TimeOut**, the server assumes the following:

```
TimeOut 600
```

This sets the timeout to 10 minutes.

Example

The following directive sets the timeout to 20 minutes, a useful time for large files on slow networks:

```
TimeOut 1200
```

AccessConfig

The **AccessConfig** directive specifies the location of the Global Access Configuration file (ACF).

Syntax

```
AccessConfig file
```

`file` is the name of the Global ACF, either a full path name or a partial path name relative to the **ServerRoot**.

You can use only one **AccessConfig** directive in the configuration file.

Default

If you do not specify a **AccessConfig** file, the server assumes the following:

```
AccessConfig conf/access.cfg
```

Examples

- The following directive instructs the server to look for the Global ACF in the file `conf/access-global` in the **ServerRoot** directory:

  ```
  AccessConfig conf/access-global
  ```

- The following directive instruct the server to look for the Global ACF in the file `/c/admin/access`:

  ```
  AccessConfig /c/admin/access
  ```

ResourceConfig

The **ResourceConfig** directive specifies the location of the Server Resource Map Configuration file.

Syntax

```
ResourceConfig file
```

`file` is the name of the Server Resource Map Configuration file, either a full path name or a partial path name relative to the **ServerRoot**.

You can use only one **ResourceConfig** directive in the configuration file.

Default

If you do not specify a **ResourceConfig** file, the server assumes the following:

```
ResourceConfig conf/srm.cfg
```

Examples

- The following directive instructs the server to look for the Server Resource Map Configuration file in the file `conf/resources` in the **ServerRoot** directory:

  ```
  ResourceConfig conf/resources
  ```

- The following directive instruct the server to look for the Server Resource Map Configuration file in the file `/c/admin/resources`:

  ```
  ResourceConfig /c/admin/resources
  ```

TypesConfig

The **TypesConfig** directive specifies the location of the typing configuration file. This file determines how the server maps file name extensions to MIME types for HTTP/1.0 clients. You should not need to edit this directive.

Syntax

```
TypesConfig file
```

`file` is the name of the Server Resource Map Configuration file, either a full path name or a partial path name relative to the **ServerRoot**.

You can use only one **TypesConfig** directive in the configuration file.

Default

If you do not specify a **TypesConfig** file, the server assumes the following:

```
TypesConfig conf/mime.typ
```

Examples

- The following directive instructs the server to look for the types configuration in the file `conf/mimetypes` in the **ServerRoot** directory:

  ```
  TypesConfig conf/mimetypes
  ```

- The following directive instructs the server to look for the types configuration in the file `/c/admin/mimetypes`:

  ```
  TypesConfig /c/admin/mimetypes
  ```

CGICacheTime

The **CGICacheTime** directive instructs the server how long a CGI Read Request VI waits before it times out.

Syntax

```
CGICacheTime time
```

`time` is the number of seconds the CGI Read Request VI waits for a request before it times out. A `time` of `-1` indicates to never time out.

You can use only one **CGICacheTime** directive in the configuration file.

Default

If you do not specify a **CGICacheTime**, the server assumes the following:

```
CGICacheTime 60
```

Examples

- To instruct the CGI Read Request VI to wait 1 minute for an incoming request before timing out, type in the following command:

  ```
  CGICacheTime 60
  ```

- To instruct the CGI Read Request VI to never time out, type in the following command. Any CGIs waiting with this timeout remain in memory until the server stops.

  ```
  CGICacheTime –1
  ```

ErrorLog

The **ErrorLog** directive determines the file to which the server logs errors and status messages it encounters. This directive logs the following information:

- Messages for Start-Up and Halt
- Clients that time out and/or abort
- Access files that attempt to override unauthorized directives
- Problems with the server
- Problems with User Authentication configuration
- Error messages stating a File does not exist

Syntax

```
ErrorLog file
```

`file` is the name of the Server Resource Map Configuration file, either a full path name or a partial path name relative to the **ServerRoot**.

You can use only one **ErrorLog** directive in the configuration file.

Default

If you do not specify an **ErrorLog**, the server assumes the following:

```
ErrorLog logs/error.log
```

Examples

- The following directive logs errors to the file `logs/errors` in the **ServerRoot** directory:

  ```
  ErrorLog logs/errors
  ```

- The following directive logs errors to the file `/tmp/errors`:

  ```
  ErrorLog /tmp/errors
  ```

- The following directive turns off error logging:

  ```
  ErrorLog /dev/null
  ```

TransferLog

The **TransferLog** directive determines where the server records client accesses.

Each line of the **TransferLog** logfile format consists of the following code:

```
host - authuser [DD/Mon/YYYY:hh:mm:ss sdddd] "request" ddd
bbbb "opt_referer" "opt_agent"
```

- `host`—DNS name or IP number of the remote client
- `authuser`—User name, if sent for authentication, otherwise the symbol (`-`)
- `DD`—Day
- `Mon`—Month (calendar name)
- `YYYY`—Year
- `hh`—Hour (24-hour format according to the machine timezone)
- `mm`—Minutes
- `ss`—Seconds
- `sdddd`—Time offset from GMT, written as a sign (`+/-`) followed by 4 digits
- `request`—First line of the HTTP request sent by the client
- `ddd`—Status code returned by the server or (`-`) if not available
- `bbbb`—Total number of bytes sent, not including the HTTP/1.0 header or (`-`) if not available
- `opt_referer`—Referer field if supplied and if **LogOptions** is `Combined`
- `opt_agent`—User agent field if supplied and if **LogOptions** is `Combined`

You can determine the name of the file accessed through `request`.

Syntax

```
TransferLog file
```

`file` is the name of the Server Resource Map Configuration file, either a full path name or a partial path name relative to the **ServerRoot**.

You can use only one **TransferLog** directive in the configuration file.

Default

If you do not specify a **TransferLog**, the server assumes the following:

```
TransferLog logs/access.log
```

Examples

- The following directive logs the transfers to the file `logs/transfers` in the **ServerRoot** directory:

  ```
  TransferLog logs/tranfers
  ```

- The following directive logs the transfers to the file `/tmp/transfers`:

  ```
  TransferLog /tmp/transfers
  ```

- The following directive turns off transfer logging:

  ```
  TransferLog /dev/null
  ```

AgentLog

The **AgentLog** directive determines where the server records client software. The **UserAgent** header, if present, is recorded in the format sent by the client. If **LogOptions** is set to `Combined`, this directive does nothing.

All clients can use this directive for statistical purposes and the tracking of protocol violations.The first whitespace-delimited word must be the software product name, with an optional slash and version designator. Other products which form part of the user agent can be written as separate words.

User Agent Log Example

The timestamp has the same format as in `access.log`.

```
[02/Oct/1996:14:48:07 -0500] Mozilla/2.0 (Macintosh; I; 68K)
[02/Oct/1996:15:01:29 -0500] Mozilla/3.0Gold (Win95; I)
[02/Oct/1996:18:23:51 -0500] Mozilla/2.0 (compatible; MSIE 3.0;
 Windows 95)
[03/Oct/1996:15:47:17 -0500]  Lynx/2.3.7 BETA  libwww/2.14
```

Syntax

```
AgentLog file
```

`file` is the name of the file to which user agents log information, either a full path name or a partial path name relative to the **ServerRoot**.

You can use only one **AgentLog** directive in the configuration file.

Default

If you do not specify an **AgentLog**, the server assumes the following:

```
AgentLog logs/agent.log
```

Examples

- The following directive logs user agents to the file `logs/agents` in the **ServerRoot** directory:

  ```
  AgentLog logs/agents
  ```

- The following directive logs user agents to the file `/tmp/agents`:

  ```
  AgentLog /tmp/agents
  ```

- The following directive turns off user agent logging:

  ```
  AgentLog /dev/null
  ```

RefererLog

The **RefererLog** directive determines where the server records refer. The **Referer** header, if present, is recorded in the following format:

```
[Time Stamp] URI -> Document
```

where `URI` is the Universal Resource Identifier for the document that references the server, and `Document` is the virtual path to the requested document. If **LogOptions** is set to `Combined`, this directive does nothing.

User RefererLog Example

The timestamp has the same format as in `access.log`.

```
[02/Oct/1996:18:22:08 -0500] http://cerberus/htdocs/ ->
/htdocs/images/logo.gif

[02/Oct/1996:18:22:11 -0500] http://cerberus/htdocs/ ->
/htdocs/config/config.htm

[02/Oct/1996:18:22:12 -0500]
http://cerberus/htdocs/config/config.htm ->
/htdocs/config/access/access.htm
```

Syntax

`RefererLog file`

`file` is the name of the file to which referers are logged, either a full path name or a partial path name relative to the **ServerRoot**.

You can use only one **RefererLog** directive in the configuration file.

Default

If you do not specify a **RefererLog**, the server assumes the following:

```
RefererLog logs/referer.log
```

Examples

- The following directive logs referers to the file `logs/referers` in the **ServerRoot** directory:

  ```
  RefererLog logs/referers
  ```

- The following directive logs referers to the file `/tmp/referers`:

  ```
  RefererLog /tmp/referers
  ```

- The following directive turns off referer logging:

  ```
  RefererLog /dev/null
  ```

☞ **Note:** `/dev/null` ***is a special path known by the server. This path also works on non-UNIX platforms.***

LogOptions

The **LogOptions** directive specifies your capability to switch between the Common Log File (CLF) format where extra information goes to separate files, or a somewhat extended CLF format where extra information is logged at the end of the normal CLF information. The **LogOptions** directive also determines your capability to change other aspects of server logging.

Because some Log Analyzers do not accept the **LogOptions** `Combined` format, you might want to check before you switch your files.

Combined Log Format Example

```
cerberus.natinst.com - - [03/Oct/1996:17:56:08 -0500] "GET /
HTTP/1.0" 200 1564 - "Mozilla/2.0 (compatible; MSIE 3.0; Windows
95)"
```

```
cerberus.natinst.com - - [03/Oct/1996:17:56:21 -0500] "GET
/images/httpsrvr.gif HTTP/1.0" 200 1032 "http://cerberus/"
"Mozilla/2.0 (compatible; MSIE 3.0; Windows 95)"
```

```
cerberus.natinst.com - - [03/Oct/1996:17:56:25 -0500] "GET
/images/lvpower.gif HTTP/1.0" 200 1686 "http://cerberus/"
"Mozilla/2.0 (compatible; MSIE 3.0; Windows 95)"
```

```
cerberus.natinst.com - - [03/Oct/1996:17:56:25 -0500] "GET
/cgi-bin/counter.vi?index.html HTTP/1.0" 200 988 "http://cerberus/"
"Mozilla/2.0 (compatible; MSIE 3.0; Windows 95)"
```

☞ **Note:** ***The code above includes extra* Refer *and* UserAgent *fields at the end of each command.***

Syntax

```
LogOption opt1 opt2
```

`optn` is one of the following:

- `Combined`—With this **LogOption**, the server does not open a **RefererLog** or **AgentLog**. Because these files are not open, the server logs the information normally logged to those files in the **TransferLog**.
- `Separate`—With this **LogOption**, the server uses separate log files for the **RefererLog** and **AgentLog**.
- `Date`—With this **LogOption**, the `Date` section of the **TransferLog** is prepended to each line of the **RefererLog** and **AgentLog** to correlate the various files. This option is only valid with `Separate`.

Default

If you do not specify **LogOption**, the server assumes the following:

```
LogOptions Separate
```

Examples

- The following directive generates separate log files with date prepended:

```
LogOptions Date
```

- The following directive generates a single log file containing all the information:

```
LogOptions Combined
```

PanelImageType

The **PanelImageType** directive specifies the default type of front panel images. Use this directive when no type is specified on a `.snap` or `.monitor` URL.

Syntax

```
PanelImageType type
```

`type` is the image type, either JPEG or PNG.

You can use only one **PanelImageType** directive in the configuration file.

Default

If you do not specify **PanelImageType**, the server assumes the following:

```
PanelImageType JPEG
```

Example

The following directive sets the default image type of front panels to PNG:

```
PanelImageType PNG
```

PanelImageDepth

The **PanelImageDepth** directive specifies the default color depth of front panel images. Use this directive when no depth is specified on a `.snap` or `.monitor` URL.

Syntax

```
PanelImageDepth depth
```

`depth` is the color depth of the image. `depth` can be 1, 4, 8, or 24 bits.

You can use only one **PanelImageDepth** directive in the configuration file.

Default

If you do not specify **PanelImageDepth**, the server assumes the following:

```
PanelImageDepth 8
```

Example

The following directive sets the default color depth of front panels images to 24 (millions of colors):

```
PanelImageDepth 24
```

PanelImageQuality

The **PanelImageQuality** directive specifies the default quality of front panel images. This only might apply to certain image types, such as JPEG. Use this directive when no quality is specified on a `.snap` or `.monitor` URL.

Syntax

```
PanelImageQuality quality
```

`quality` is the quality of the image in percent (0–100, with 100 being best).

You can use only one **PanelImageQuality** directive in the configuration file.

Default

If you do not specify **PanelImageQuality**, the server assumes the following:

```
PanelImageQuality 80
```

Example

The following directive sets the default quality of front panels images to 50%:

```
PanelImageQuality 50
```

PanelImageCacheCompactTime

The **PanelImageCacheCompactTime** directive specifies how often the cache of panel images is compacted.

Syntax

```
PanelImageCacheCompactTime time
```

`time` is number of seconds after which the cache is compacted.

You can use only one **PanelImageCacheCompactTime** directive in the configuration file.

Default

If you do not specify **PanelImageCacheCompactTime**, the server assumes the following:

```
PanelImageCacheCompactTime 300
```

Example

The following directive compacts the panel image cache every 10 minutes:

```
PanelImageCacheCompactTime 600
```

ServerPushRefresh

The **ServerPushRefresh** directive specifies the default refresh rate for server-push animations. This value is used when no refresh is specified on a `.spool` or `.monitor` URL.

Syntax

```
ServerPushRefresh time
```

`time` is the number of seconds between each image.

You can use only one **ServerPushRefresh** directive in the configuration file.

Default

If you do not specify **ServerPushRefresh**, the server assumes the following:

```
ServerPushRefresh 10.0
```

Example

The following directive sets the default server-push refresh rate at once every 5 seconds:

```
ServerPushRefresh 5.0
```

ServerPushLifespan

The **ServerPushLifespan** directive specifies the lifespan of server-push animations. This value is used when no lifespan is specified on a `.spool` or `.monitor` URL.

Syntax

```
ServerPushLifespan time
```

`time` is the number of seconds before a server-push animation is terminated. A negative number indicates that only the client can stop the animation.

You can use only one **ServerPushLifespan** directive in the configuration file.

Default

If you do not specify **ServerPushLifespan**, the server assumes the following:

```
ServerPushLifespan 300
```

Examples

- The following directive sets the default server-push length of animation to 1 minute:

  ```
  ServerPushLifespan 60
  ```

- The following directive sets the default server-push length of animation to 0 seconds, which returns a static image:

  ```
  ServerPushLifespan 0
  ```

- The following directive sets the default server-push length of animation to infinity, which causes the server to wait for the client to stop the animation:

  ```
  ServerPushLifespan –1
  ```

ServerPushMaxConnections

The **ServerPushMaxConnections** directive specifies the maximum number of server-push animations the server returns at a time. When **ServerPushMaxConnections** server-push animations are running and a new request comes in, the animation older than **ServerPushMinTime** seconds stops and the new animation starts. If none of the animations are older than **ServerPushMinTime**, the servers returns a static image for this request.

Syntax

```
ServerPushMaxConnections num
```

`num` is the maximum number of parallel server-push connections.

You can use only one **ServerPushMaxConnections** directive in the configuration file.

Default

If you do not specify **ServerPushMaxConnections**, the server assumes the following:

```
ServerPushMaxConnections 40
```

Example

The following directive sets the maximum number of parallel server-push animations to 10:

```
ServerPushMaxConnections 10
```

ServerPushMinTime

The **ServerPushMinTime** directive specifies the minimum number of seconds a server-push animation must exist before the server can stop it prematurely in order to service another server-push request. When **ServerPushMaxConnections** server-push animations are running and a new request comes in, the animation older than **ServerPushMinTime** seconds stops and the new animation starts. If none of the animations are older than **ServerPushMinTime**, the server returns a static image for this request.

Syntax

```
ServerPushMinTime time
```

`time` is the number of seconds before which the server can stop a server-push animation.

You can use only one **ServerPushMinTime** directive in the configuration file.

Default

If you do not specify **ServerPushMinTime**, the server assumes the following:

```
ServerPushMinTime 240
```

Example

The following directive sets the time before which a server-push animation can be prematurely stopped to 1 minute:

```
ServerPushMinTime 60
```

Server Resource Map Configuration Directives

The Server Resource Map Configuration directives control the presentation of the server documents. For more information on the Server Resource Map Configuration specification, refer to the section *Server Resource Map Configuration*, in Chapter 6, *G Web Server*.

DocumentRoot

The **DocumentRoot** directive determines the directory from which the server serves files.

If you want to serve files outside this directory, you can use the **Alias** directive.

Syntax

`DocumentRoot dir`

`dir` is the absolute path of the directory you want documents to be served from. The path is in UNIX format, which uses the path separator `/`.

You can use only one **DocumentRoot** directive in the configuration file.

File Name

`srm.cfg`

Default

If you do not specify a **DocumentRoot**, the server assumes the following:

```
DocumentRoot internet/http/htdocs
```

Examples

- The following directive sets the **DocumentRoot** to the directory `C:\LABVIEW\HOME` on Windows:

  ```
  DocumentRoot /c/labview/home
  ```

- The following directive sets the **DocumentRoot** to the directory `Macintosh HD:LabVIEW Folder:Home` on the Mac:

  ```
  DocumentRoot /Macintosh\ HD/LabVIEW\ Folder/Home
  ```

AccessFileName

The **AccesFileName** directive determines the name of the file the server checks to find directory Access Configuration files (ACFs). When returning documents to a client, the server looks for ACFs in the document directory as well as the parent directories.

Syntax

```
AccessFileName file
```

`file` is a file name.

You can use only one **AccessFileName** directive in the configuration file.

File Name

`srm.cfg`

Default

If you do not specify an **AccessFileName**, the server assumes the following:

```
AccessFileName htaccess.txt
```

Example

The following directive sets the **AccessFileName** to `.htaccess`:

```
AccessFileName .htaccess
```

Redirect

The **Redirect** directive creates a virtual document on your server and redirects any access to the document to a new URL.

Syntax

```
Redirect virtual URL
```

`virtual` is the translated location which triggers a redirect.

`URL` is the URL of the new document.

You can use several **Redirect** directives in the configuration file.

File Name

`srm.cfg`

Default

There are no default **Redirect** directives.

Example

The following directive requests the document `/files` be redirected to the new location, `ftp://ftp.natinst.com`:

```
Redirect /files ftp://ftp.natinst.com
```

Alias

The **Alias** directive creates a virtual document or directory on your server and uses a different file or directory name to satisfy access.

Syntax

```
Alias virtual path
```

`virtual` is the translated location of the file or directory.

`path` is the path name of the file or directory used to fulfill the request. If `path` is a relative path, it is appended to **DocumentRoot**.

You can use several **Alias** directives in the configuration file.

File Name

`srm.cfg`

Default

By default, the server contains the following two **Alias** directives, which you can override:

```
Alias /htdocs/ <ServerRoot>/htdocs/
Alias /icons/ <ServerRoot>/icons/
```

Examples

- The following directive generates requests for `/images` from the directory `/ftp/pub/images`:

  ```
  Alias /images /ftp/pub/images
  ```

- The following directive generates requests for `/docs/` from the directory `<DocumentRoot>/documents/`:

  ```
  Alias /docs/ documents/
  ```

ScriptAlias

The **ScriptAlias** directive creates a virtual directory on your server. You satisfy access to the VIs in that virtual directory by returning the output of the CGI VI.

Syntax

`ScriptAlias virtual path`

`virtual` is the translated location of the CGI directory.

`path` is the path name of the directory containing the CGI VIs that fulfill the request. If `path` is a relative path, it is appended to **DocumentRoot**.

You can use several **ScriptAlias** directives in the configuration file.

File Name

`srm.cfg`

Default

There are no default **ScriptAlias** directives.

Examples

- The following directive fulfills the request `/cgi-bin/foo.vi` by running the VI `C:\labview\cgi-bin\foo.vi` on Windows:

  ```
  ScriptAlias /cgi-bin/ /c/labview/cgi-bin/
  ```

- The following directive fulfills the request `/cgi-bin/foo.vi` by running the VI `<DocumentRoot>/cgi-bin/foo.vi`:

  ```
  ScriptAlias /cgi-bin/ cgi-bin/
  ```

AddType

The **AddType** directive adds entries to the server default typing information and generates a particular extension type. This directives overrides any conflicting entries in the **TypesConfig** file.

Syntax

```
AddType type/subtype extension
```

`type/subtype` is the MIME-like type for the document.

`extension` is the file name extension to map to this type.

You can use several **AddType** directives in the configuration file.

File Name

`srm.cfg`

Default

The default types are located in the types configuration files.

Examples

- The following directive serves any file ending in `.doc` as type `text/plain`:

  ```
  AddType text/plain doc
  ```

- The following directive serves any file ending in `.txt` as type `text/plain`:

  ```
  AddType text/plain txt
  ```

AddEncoding

The **AddEncoding** directive specifies an encoding type for a document with a given extension.

In order to serve encoded documents to clients, the client must work with the given encoding method, as well as the HTTP encoding extension.

Syntax

```
AddEncoding type extension
```

`type` is the encoding type for the document.

`extension` is the file name extension to map to this encoding.

You can use several **AddEncoding** directives in the configuration file.

File Name

`srm.cfg`

Default

There are no default encodings.

Example

The following directive marks any file ending in `.gz` as encoded using the `x-gzip` method:

```
AddEncoding x-gzip gz
```

Reference

Refer to the HTTP/1.0 Section 3.5 Content Codings.

DefaultType

The **DefaultType** directive maps a file to a default type if the server cannot type a file through normal means.

Syntax

```
DefaultType type/subtype
```

`type/subtype` is the MIME-like type.

You can use only one **DefaultType** directive in the configuration file.

File Name

`srm.cfg`

Default

If no **DefaultType** is present, the server assumes the following:

```
DefaultType text/html
```

Example

The following directive returns files with an unknown extension type `application/octet-stream` (generic binary file):

```
DefaultType application/octet-stream
```

DirectoryIndex

The **DirectoryIndex** directive determines the file the server looks for to generate a pre-written index to a given directory.

Syntax

`DiretoryIndex file`

`file` is a file name.

You can use only one **DirectoryIndex** directive in the configuration file.

File Name

`srm.cfg`

Default

If no **DirectoryIndex** is present, the server assumes the following:

```
DirectoryIndex index.htm
```

Example

The following directive sets **DirectoryIndex** to `dir.htm`. This request for `/dir/` instructs the server to look for the file `DocumentRoot/dir/dir.htm`. If found, the server sends the file back to the client. Otherwise, the server creates and returns an index from the file system.

```
DirectoryIndex dir.htm
```

FancyIndexing

The **FancyIndexing** directive specifies whether you want fancy directory indexing including icons and file sizes or standard indexing for your file.

Turning fancy indexing off improves the performance of your server, especially on a slow file system.

Syntax

`FancyIndexing setting`

`setting` is either ON or OFF.

You can use only one **FancyIndexing** directive in the configuration file.

File Name

`srm.cfg`

Default

If no **FancyIndexing** is present, the server assumes the following:

```
FancyIndexing off
```

Example

The following directive turns on fancy indexing:

```
FancyIndexing on
```

DefaultIcon

The **DefaultIcon** directive specifies what icon is shown in automatically generated directory listings for a file containing no icon information.

Syntax

`DefaultIcon location`

`location` is the virtual path to the icon on your server.

You can use only one **DefaultIcon** directive in the configuration file.

File Name

`srm.cfg`

Default

If no **DefaultIcon** is present, the server assumes nothing.

Example

The following directive gives a file with no icon the icon `/icons/unknown.gif`:

```
DefaultIcon /icons/unknown.gif
```

ReadmeName

The **ReadmeName** directive specifies what file name the server looks for when indexing a directory. This directive adds a description paragraph to the end of the index it automatically generates.

Syntax

```
ReadmeName name
```

`name` is the name of the file the server looks for when trying to find a description file. The server first looks for `name.html` and `name.htm`. If found, the server displays the HTML inlined with its own index. If the server finds `name`, it includes the file as plaintext.

You can use only one **ReadmeName** directive in the configuration file.

File Name

`srm.cfg`

Default

If no **ReadmeName** is present, the server assumes nothing.

Example

The following directive generates an index for the directory `/foo/`, using the `name` files `/foo/README.html`, `/foo/README.htm`, and `/foo/README`:

```
ReadmeName README
```

HeaderName

The **HeaderName** directive determines what file name the server looks for when indexing a directory with a custom header.

Syntax

```
HeaderName name
```

name is the name of the file the server looks for when trying to find a description file. The server first looks for `name.html` and `name.htm`. If found, the server displays the HTML inlined with its own index. If the server finds name, it includes the file as plaintext.

You can use only one **HeaderName** directive in the configuration file.

File Name

`srm.cfg`

Default

If no **HeaderName** is present, the server assumes nothing.

Example

The following directive generates an index for the directory `/foo/`, using the name files `/foo/HEADER.html`, `/foo/HEADER.htm`, and `/foo/HEADER`:

```
HeaderName HEADER
```

AddIcon

The **AddIcon** directive determines kind of icon the server uses to represent a given file type in a directory index.

Syntax

```
AddIcon icon name1 name2...
```

icon is a virtual path to an image file which is shown for files which match the pattern of names. Alternatively, this can be a group of the format `(alt,icon)` where `alt` is the 3-letter text tag given for an icon for non-graphical browsers, and `icon` is a virtual path.

name is either `^^DIRECTORY^^` for directories, `^^BLANKICON^^` specifying the blank icon used to format the list properly, a file extension (`.html`), or a wildcard expression.

You can use several **AddIcon** directives in the configuration file.

File Name

`srm.cfg`

Default

There are no default icons.

Examples

- The following directive references the image `/icons/image.gif` for display next to the file name when indexing a file with the extensions `.gif`, `.jpg`, or `.xbm`:

  ```
  AddIcon /icons/image.gif .gif .jpg .xbm
  ```

- The following directive references `/icons/dir.gif` as the image for a subdirectory:

  ```
  AddIcon /icons/dir.gif ^^DIRECTORY^^
  ```

- The following directive references `/icons/sound.gif` as the image to show next to any `.au` sound file, with the textual `ALT` tag of `SND` for non-image clients:

  ```
  AddIcon (SND,/icons/sound.gif) *.au
  ```

AddIconByType

The **AddIconByType** directive determines the kind of icon to show for a given file type in a directory index.

Syntax

```
AddIcon icon type1 type2...
```

`icon` is a virtual path to an image file shown for files which match the pattern of names. Alternatively, this can be a group of the format `(alt,icon)` where `alt` is the three-letter text tag given for an icon for non-graphical browsers, and `icon` is a virtual path.

`name` is a wildcard expression of the MIME-types to add to this icon.

You can use several **AddIconByType** directives in the configuration file.

File Name

`srm.cfg`

Default

There are no default icons.

Examples

- The following directive references `/icons/image.gif` as the image to show next to the file name when the server finds a image file:

  ```
  AddIconByType /icons/image.gif image/*
  ```

- The following directive references `/icons/sound.gif` as the image to show next to any sound file, with the textual `ALT` tag of `SND` for non-image clients:

  ```
  AddIconByType (SND,/icons/sound.gif) audio/*
  ```

AddIconByEncoding

The **AddIconByEncoding** directive determines the kind of icon to show for a given file type in a directory index.

Syntax

```
AddIconByEncoding icon name1 name2...
```

`icon` is a virtual path to an image file shown for files which match the pattern of names. Alternatively, this can be a group of the format `(alt,icon)` where `alt` is the three-letter text tag given for an icon for non-graphical browsers, and `icon` is a virtual path.

`name` is a wildcard expression of the content-encoding to add to this icon.

You can use several **AddIconByEncoding** directives in the configuration file.

File Name

`srm.cfg`

Default

There are no default icons.

Examples

- The following directive references `/icons/compress.gif` as the image to show next to the file name if a file is compressed with the UNIX compress command:

  ```
  AddIconByEncoding /icons/compress.gif x-compress
  ```

- The following directive references `/icons/compress.gif` as the image to show next to UNIX compressed files, with the textual `ALT` tag of `CMP` for non-image clients:

  ```
  AddIconByEncoding (CMP,/icons/compress.gif) x-compress
  ```

IndexIgnore

The **IndexIgnore** directive specifies which files the server ignores when generating a directory index.

Syntax

```
IndexIgnore pat1 pat2...
```

`pat` is a wildcard expression against which files are matched.

You can use several **IndexIgnore** directives in the configuration file.

File Name

`srm.cfg`

Default

The only entry ignored by default is the current directory.

Example

The following directive instructs the server to ignore files named `README`, `README.html` and any file that starts with a period:

```
IndexIgnore */README */README.htm */.*
```

ErrorDocument

The **ErrorDocument** directive points the server to a file to send instead of the built-in error message.

Syntax

```
ErrorDocument type filename
```

`type` is one of the following:

- `302—REDIRECT`
- `400—BAD_REQUEST`
- `401—AUTH_REQUIRED`
- `403—FORBIDDEN`
- `404—NOT_FOUND`
- `500—SERVER_ERROR`
- `501—NOT_IMPLEMENTED`

`filename` is a path to a `text/html` file. If `filename` is relative, it is appended to the **DocumentRoot**.

You can use only one **ErrorDocument** directive in the configuration file.

File Name

`srm.cfg`

Default

By default, the compiled error messages are used.

Examples

- `ErrorDocument 403 /error/fobidden.htm`
- `ErrorDocument 500 /error/srvrerr.htm`

Error Codes

Appendix B

This appendix lists the error codes returned by the Internet Developers Toolkit for G VIs, including the error number and a description. Each VI returns an error code that indicates whether the function was performed successfully. You also can find this appendix in your Online Help reference.

Table B-1. Error Codes

Code	Type	Description
15110	FTP	110—Restart marker reply. In this case, the text is exact and not left to the particular implementation; it must read: `MARK yyyy = mmmm` where `yyyy` is the user-process data stream marker, and `mmmm` is the server equivalent marker (note the spaces between markers and =).
15120	FTP	120—Service ready in `nnn` minutes.
15125	FTP	125—Data connection already open; transfer starting.
15150	FTP	150—File status OK; about to open data connection.
15200	FTP	200—Command OK.
15202	FTP	202—Command not implemented; superfluous at this site.
15211	FTP	211—System status or system help reply.
15212	FTP	212—Directory status.
15213	FTP	213—File status.
15214	FTP	214—Help message; also how to use the server or the meaning of a particular non-standard command. This reply is only useful to the human user.
15215	FTP	215—`NAME` system type, where `NAME` is an official system name from the list in the Assigned Numbers document.
15220	FTP	220—Service ready for new user.
15221	FTP	221—Service closing control connection. Logged out if appropriate.
15225	FTP	225—Data connection open; no transfer in progress.
15226	FTP	226—Closing data connection; requested file action successful.
15227	FTP	227—Entering Passive Mode (`h1,h2,h3,h4,p1,p2`).
15230	FTP	230—User logged in; proceed.

Table B-1. Error Codes (Continued)

Code	Type	Description
15250	FTP	250—Requested file action OK; completed.
15257	FTP	257—`PATHNAME` created.
15331	FTP	331—User name OK; need password.
15332	FTP	332—Need account for login.
15350	FTP	350—Requested file action pending further information.
15421	FTP	421—Service not available; closing control connection. This can be a reply to any command if the service knows it must shut down.
15425	FTP	425—Cannot open data connection.
15426	FTP	426—Connection closed; transfer aborted.
15450	FTP	450—Requested file action not taken; file unavailable.
15451	FTP	451—Requested action aborted; local error in processing.
15452	FTP	452—Requested action not taken; insufficient storage space in system.
15500	FTP	500—Syntax error; command unrecognized. This can include errors such as command line too long.
15501	FTP	501—Syntax error in parameters or arguments.
15502	FTP	502—Command not implemented.
15503	FTP	503—Bad sequence of commands.
15504	FTP	504—Command not implemented for that parameter.
15530	FTP	530—Not logged in.
15532	FTP	532—Need account for storing files.
15550	FTP	550—Requested action not taken; file unavailable.
15551	FTP	551—Requested action aborted; page type unknown.
15552	FTP	552—Requested file action aborted; exceeded storage allocation (for current directory or dataset).
15553	FTP	553—Requested action not taken; file name not allowed.
16211	SMTP	211—System status or system help reply.
16214	SMTP	214—Help message. Provides information on how to use the receiver or the meaning of a particular non-standard command. This reply is only useful to the human user.
16220	SMTP	220—`<domain>` Service ready.
16221	SMTP	221—`<domain>` Service closing transmission channel.

Table B-1. Error Codes (Continued)

Code	Type	Description
16250	SMTP	250—Requested mail action OK; completed.
16251	SMTP	251—User not local; will forward to `<forward-path>`.
16354	SMTP	354—Start mail input; end with `<CRLF>.<CRLF>`.
16421	SMTP	421—`<domain>` Service not available; closing transmission channel. This can be a reply to any command if the service knows it must shut down.
16450	SMTP	450—Requested mail action not taken; mailbox unavailable.
16451	SMTP	451—Requested action aborted; local error in processing.
16452	SMTP	452—Requested action not taken; insufficient system storage.
16500	SMTP	500—Syntax error; command unrecognized. This can include errors such as command line too long.
16501	SMTP	501—Syntax error in parameters or arguments.
16502	SMTP	502—Command not implemented.
16503	SMTP	503—Bad sequence of commands.
16504	SMTP	504—Command parameter not implemented.
16550	SMTP	550—Requested action not taken; mailbox unavailable.
16551	SMTP	551—User not local; please try `<forward-path>`.
16552	SMTP	552—Requested mail action aborted; exceeded storage allocation.
16553	SMTP	553—Requested action not taken; mailbox name not allowed.
16554	SMTP	554—Transaction failed.
17200	HTTP	200—OK. The request has succeeded. The information returned with the response is dependent on the method used in the request.
17201	HTTP	201—Created. The request has been fulfilled and resulted in a new resource being created.
17202	HTTP	202—Accepted. The request has been accepted for processing, but the processing has not been completed.
17204	HTTP	204—No Content. The server has fulfilled the request but there is no new information to send back.
17300	HTTP	300—Multiple Choices. The requested resource is available at one or more locations.
17301	HTTP	301—Moved Permanently. The requested resource has been assigned a new permanent URL and any future references to this resource should be done using that URL.

Table B-1. Error Codes (Continued)

Code	Type	Description
17302	HTTP	302—Moved Temporarily. The requested resource resides temporarily under a different URL.
17304	HTTP	304—Not Modified. If the client has performed a conditional `GET` request and access is allowed, but the document has not been modified since the date and time specified in the `If-Modified-Since` field.
17400	HTTP	400—Bad Request. The server could not understand the request because of malformed syntax.
17401	HTTP	401—Unauthorized. The request requires user authentication.
17403	HTTP	403—Forbidden. The server understood the request, but is refusing to fulfill it.
17404	HTTP	404—Not Found. The server has not found anything matching the `Request-URL`.
17500	HTTP	500—Internal Server Error. The server encountered an unexpected condition which prevented it from fulfilling the request.
17501	HTTP	501—Not Implemented. The server does not work with the functionality required to fulfill the request.
17502	HTTP	502—Bad Gateway. The server, while acting as a gateway or proxy, received an invalid response from the upstream server it accessed in attempting to fulfill the request.
17503	HTTP	503—Service Unavailable. The server is currently unable to handle the request due to a temporary overloading or maintenance of the server.
18000	URL	Invalid URL protocol.

Customer Communication

Appendix C

For your convenience, this appendix contains forms to help you gather the information necessary to help us solve your technical problems and a form you can use to comment on the product documentation. When you contact us, we need the information on the Technical Support Form and the configuration form, if your manual contains one, about your system configuration to answer your questions as quickly as possible.

National Instruments has technical assistance through electronic, fax, and telephone systems to quickly provide the information you need. Our electronic services include a bulletin board service, an FTP site, a fax-on-demand system, and e-mail support. If you have a hardware or software problem, first try the electronic support systems. If the information available on these systems does not answer your questions, we offer fax and telephone support through our technical support centers, which are staffed by applications engineers.

Electronic Services

Bulletin Board Support

National Instruments has BBS and FTP sites dedicated for 24-hour support with a collection of files and documents to answer most common customer questions. From these sites, you can also download the latest instrument drivers, updates, and example programs. For recorded instructions on how to use the bulletin board and FTP services and for BBS automated information, call (512) 795-6990. You can access these services at:

United States: (512) 794-5422
Up to 14,400 baud, 8 data bits, 1 stop bit, no parity

United Kingdom: 01635 551422
Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

France: 01 48 65 15 59
Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

FTP Support

To access our FTP site, log on to our Internet host, `ftp.natinst.com`, as `anonymous` and use your Internet address, such as `joesmith@anywhere.com`, as your password. The support files and documents are located in the `/support` directories.

Fax-on-Demand Support

Fax-on-Demand is a 24-hour information retrieval system containing a library of documents on a wide range of technical information. You can access Fax-on-Demand from a touch-tone telephone at (512) 418-1111.

E-Mail Support (currently U.S. only)

You can submit technical support questions to the applications engineering team through e-mail at the Internet address listed below. Remember to include your name, address, and phone number so we can contact you with solutions and suggestions.

`support@natinst.com`

Telephone and Fax Support

National Instruments has branch offices all over the world. Use the list below to find the technical support number for your country. If there is no National Instruments office in your country, contact the source from which you purchased your software to obtain support.

	Telephone	Fax
Australia	02 9874 4100	02 9874 4455
Austria	0662 45 79 90 0	0662 45 79 90 19
Belgium	02 757 00 20	02 757 03 11
Canada (Ontario)	905 785 0085	905 785 0086
Canada (Quebec)	514 694 8521	514 694 4399
Denmark	45 76 26 00	45 76 26 02
Finland	09 527 2321	09 502 2930
France	01 48 14 24 24	01 48 14 24 14
Germany	089 741 31 30	089 714 60 35
Hong Kong	2645 3186	2686 8505
Israel	03 5734815	03 5734816
Italy	02 413091	02 41309215
Japan	03 5472 2970	03 5472 2977
Korea	02 596 7456	02 596 7455
Mexico	5 520 2635	5 520 3282
Netherlands	0348 433466	0348 430673
Norway	32 84 84 00	32 84 86 00
Singapore	2265886	2265887
Spain	91 640 0085	91 640 0533
Sweden	08 730 49 70	08 730 43 70
Switzerland	056 200 51 51	056 200 51 55
Taiwan	02 377 1200	02 737 4644
U.K.	01635 523545	01635 523154

Technical Support Form

Photocopy this form and update it each time you make changes to your software or hardware, and use the completed copy of this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

If you are using any National Instruments hardware or software products related to this problem, include the configuration forms from their user manuals. Include additional pages if necessary.

Name _____

Company _____

Address _____

_____

Fax (___)_____ Phone (___) _____

Computer brand _____ Model _____ Processor_____

Operating system (include version number) _____

Clock speed _____MHz RAM _____MB Display adapter _____

Mouse ___yes ___no Other adapters installed _____

Hard disk capacity _____MB Brand _____

Instruments used _____

_____

National Instruments hardware product model_____ Revision _____

Configuration _____

National Instruments software product _____ Version _____

Configuration _____

The problem is: _____

_____

_____

_____

_____

List any error messages: _____

_____

_____

The following steps reproduce the problem:_____

_____

_____

_____

_____

Internet Developers Toolkit for G
Hardware and Software Configuration Form

Record the settings and revisions of your hardware and software on the line to the right of each item. Complete a new copy of this form each time you revise your software or hardware configuration, and use this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

National Instruments Products

DAQ hardware _____

Interrupt level of hardware _____

DMA channels of hardware _____

Base I/O address of hardware _____

Programming choice _____

BridgeVIEW or LabVIEW version _____

Other boards in system _____

Base I/O address of other boards _____

DMA channels of other boards _____

Interrupt level of other boards _____

Other Products

Computer make and model _____

Microprocessor _____

Clock frequency or speed _____

Type of video board installed _____

Operating system version _____

Operating system mode _____

Programming language _____

Programming language version _____

Other boards in system _____

Base I/O address of other boards _____

DMA channels of other boards _____

Interrupt level of other boards _____

Documentation Comment Form

National Instruments encourages you to comment on the documentation supplied with our products. This information helps us provide quality products to meet your needs.

Title: *Internet Developers Toolkit for G Reference Manual*

Edition Date: January 1997

Part Number: 321392A-01

Please comment on the completeness, clarity, and organization of the manual.

_____

_____

_____

_____

_____

_____

_____

If you find errors in the manual, please record the page numbers and describe the errors.

_____

_____

_____

_____

_____

_____

_____

Thank you for your help.

Name _____

Title _____

Company _____

Address _____

_____

Phone (___)_____ Fax (___) _____

Mail to: Technical Publications
National Instruments Corporation
6504 Bridge Point Parkway
Austin, TX 78730-5039

Fax to: Technical Publications
National Instruments Corporation
(512) 794-5678

Glossary

A

ACF	Access Configuration File.
active FTP mode	A File Transfer Protocol (FTP) mode in which the FTP server connects back to the FTP client to make data connections; this default mode of operation might not work from inside strict Internet firewalls.
applet	An embedded Java application in a HTML document.
Application Programming Interface (API)	A library of programming routines for performing a specific task.
array	Ordered, indexed set of data elements of the same type.
ASCII	American Standard Code for Information Interchange.

B

block diagram	Pictorial description or representation of a program or algorithm.
browser	Software that displays World Wide Web pages on your computer.

C

client-pull	Automatic request for a new document initiated by the client.
client-side	Documents an application residing on a browser machine.
Common Gateway Interface (CGI)	Standard for external gateway programs to interface with information servers, such as HTTP servers.
cookie	A token identifying some other information.

D

directive	A line in a configuration file, consisting of a keyword and some data, used to configure the G Web Server.
Domain Name System (DNS)	A distributed name/address database used on the Internet.

E

environment variables (FTP)	Variables, used by a CGI application, that contain information about the server and the request.

F

File Transfer Protocol	An application-level protocol used for accessing files or remote hosts on the Internet.
front panel	The interactive user interface of a VI. Modeled from the front panel of physical instruments, it is composed of switches, slides, meters, graphs, charts, gauges, LEDs, and other controls.

G

G	The BridgeVIEW and LabVIEW graphical programming language.
Graphics Interchange Format (GIF)	A proprietary file format developed by CompuServe Information Service for the storage of 8-bit color, raster (bitmap) images, based on the LZW data compression algorithm.

H

hexadecimal	A base-16 number system.
HyperText Markup Language (HTML)	Language used to create hypertext documents. HTML tags can encode text attributes as well as specifying links to documents or embedded images.
HyperText Transfer Protocol (HTTP)	A application-level protocol used to serve documents to remote clients on the Internet (World Wide Web).

I

identity transliteration file	A mapping file that does not change the text it maps.
Image Mode	The FTP transfer mode used for binary files.
IMG tag	Tag used to embed an image in an HTML document.
Internet Protocol (IP)	An internetwork layer protocol that routes datagrams between hosts on the Internet.

J

Joint Photographic Experts Group (JPEG)	A standard designed for compressing color or gray-scale images of natural, real-world scenes.

K

Keyed Array	A string array datatype in which elements are indexed by key strings.

L

LED	Light-emitting diode.

M

Multipurpose Internet Mail Extensions (MIME)	A specification for the interchange of multi-media e-mail among different computer systems that use Internet mail standards.

N

Network Virtual Terminal (NVT)	An imaginary device which provides a standard, network-wide, intermediate representation of a concanical terminal.

P

parse	To determine the syntactic structure of a sentence or string of symbols.

passive FTP mode	A File Transfer Protocol (FTP) mode in which the FTP client initiates data connections to the FTP server; while this mode of operation works inside a strict Internet firewall, the default mode is *active FTP* because passive FTP does not work with some FTP servers.
Perl	A programming language which easily manipulates text, files, and processes.
Portable Network Graphics (PNG)	An extensible file format for the lossless, portable, well-compressed storage of raster (bitmap) images.
protocol	Set of rules that a computer uses to connect with other computers, specifying the format, timing, sequencing, and error checking for data transmission.

S

sectioning directive	Directive that can include other directives which are enclosed between opening and closing directive tags.
server-push animations	Animations consisting of a series of images sent by an HTTP server over a period of time.
Simple Mail Transfer Protocol (SMTP)	Application-level protocol used for sending and forwarding e-mail on the Internet.
subVI	VI used in the block diagram of another VI; comparable to a subroutine.

T

tag	Specific HTML codes; specific word surrounded by less than (<) and greater than (>) signs.

target character set — A standard character set, such as ISO-Latin-1, used to encode an e-mail message or attachment for transmission over the Internet; a *virtual character set* is converted to a target character set by the process of *transliteration*.

Telnet — A TCP/IP protocol consisting of data and embedded control characters.

transliteration — Translating characters from one character set to another.

Transmission Control Protocol (TCP) — A transport layer protocol used on the Internet.

U

Universal Resource Locator (URL) — Form in which resources on the Internet are addressed on the World Wide Web.

V

virtual character set — A local character set used by a user to enter an e-mail message or attachment; a virtual character set is converted to a *target character set* by the process of *transliteration*.

virtual instrument (VI) — G program; so called because it models the appearance and function of a physical instrument.

W

wildcard expression — A string-matching expression in which a ? represents an arbitrary character and * represents zero or more arbitrary characters.

Index

A

Access Configuration, 6-5 to 6-6
    aspects of control, 6-5
    Global Access Configuration file, 6-5
    list of directives, 6-6
    Per-Directory Access Configuration file, 6-5
    purpose, 6-5
Access Configuration directives, A-1 to A-14
    Allow, A-10 to A-11
    AllowOverride, A-3 to A-4
    AuthGroupFile, A-6 to A-7
    AuthName, A-4 to A-5
    AuthType, A-5
    AuthUserFile, A-5 to A-6
    Deny, A-9 to A-10
    Directory, A-2
    Limit, A-7 to A-8
    list of directives, 6-6
    OnDeny, A-13 to A-14
    Order, A-8 to A-9
    Panel, A-2 to A-3
    Require, A-11 to A-12
    Satisfy, A-12 to A-13
AccessConfig directive, A-18 to A-19
AccessFileName directive, A-33
Active CGI VIs, HTTP Server, 6-3
active connection for FTP, 2-1
Active Connections, HTTP Server, 6-3
AddEncoding directive, A-37
AddIcon directive, A-41 to A-42
AddIconByEncoding directive, A-43
AddIconByType directive, A-42 to A-43
AddType directive, A-36
AgentLog directive, A-23 to A-24
Alias directive, A-34 to A-35
Allow directive, A-10 to A-11
AllowOverride directive, A-3 to A-4
animated panel image
    .monitor URL, 6-14 to 6-16
        examples, 6-16
        purpose and use, 6-14 to 6-15
        syntax, 6-15 to 6-16
    .spool URL, 6-16 to 6-18
        examples, 6-17 to 6-18
        purpose and use, 6-16
        syntax, 6-17
ASCII character set, 3-3
ASCII mode for FTP, 2-2
AuthGroupFile directive, A-6 to A-7
AuthName directive, A-4 to A-5
AuthType directive, A-5
AuthUserFile directive, A-5 to A-6

B

bulletin board support, C-1

C

Cached CGI VIs, HTTP Server, 6-3
Case sensitivity, in configuration files, 6-4
CGI (Common Gateway Interface)
    definition, 6-18
CGI applications, 6-18 to 6-21
    application process (figure), 6-19
    building in G, 1-5
    CGI VI example, 6-21
    definition, 6-18
    environment variables (table), 6-20 to 6-21
    GET method of supplying parameters, 6-19 to 6-20

POST method of supplying parameters, 6-19
structure of CGI application (figure), 6-21
CGI Parameter VIs, 6-22 to 6-23
CGI Utility VIs, 6-22 to 6-32
active, 6-3
cached, 6-3
categories, 6-22
CGI Parameter VIs, 6-22 to 6-23
File Path VIs, 6-23 to 6-24
HTML VIs, 6-26 to 6-32
HTML Document VI (example), 6-29
HTML Generic Tag VI (example), 6-28
HTML table block diagram, 6-27
HTML+ Monitor VI (example), 6-28
HTML+ String Array to Table VI (example), 6-27 to 6-28
list of VIs, 6-30 to 6-32
Keyed Array VIs, 6-25 to 6-26
Network VIs, 6-24
Panel Image VIs, 6-24 to 6-25
CGICacheTime directive, A-20 to A-21
character codes, 3-3
character sets, 3-2 to 3-4
definition, 3-3
ISO Latin-1, 3-3
Macintosh, 3-4
overview, 3-2 to 3-3
transliteration, 3-4
US-ASCII, 3-3
characters, 3-2 to 3-3
client state information, maintaining, 6-32 to 6-35
client-side cookies, 6-33
Cookie VIs, 6-34 to 6-35
server-side cookies, 6-33
using cookies, 6-34 to 6-35
client-side cookies, 6-33
comment lines, in configuration files, 6-4
Common Gateway Interface (CGI). *See also* CGI applications; CGI Utility VIs.
definition, 6-18
configuration, 1-6 to 1-7
[ftp] configuration, 1-6
[http] configuration, 1-7
[smtp] configuration, 1-6 to 1-7
configuration directives, A-1 to A-45
Access Configuration, A-1 to A-14
Allow, A-10 to A-11
AllowOverride, A-3 to A-4
AuthGroupFile, A-6 to A-7
AuthName, A-4 to A-5
AuthType, A-5
AuthUserFile, A-5 to A-6
Deny, A-9 to A-10
Directory, A-2
Limit, A-7 to A-8
list of directives, 6-6
OnDeny, A-13 to A-14
Order, A-8 to A-9
Panel, A-2 to A-3
Require, A-11 to A-12
Satisfy, A-12 to A-13
in configuration files, 6-4
Server Configuration, A-14 to A-32
AccessConfig, A-18 to A-19
AgentLog, A-23 to A-24
CGICacheTime, A-20 to A-21
ErrorLog, A-21 to A-22
list of directives, 6-6 to 6-7
LogOptions, A-26 to A-27
PanelImageCacheCompactTime, A-29
PanelImageDepth, A-28
PanelImageQuality, A-28 to A-29
PanelImageType, A-27
Port, A-14 to A-15
RefererLog, A-24 to A-25
ResourceConfig, A-19
ServerAdmin, A-15 to A-16
ServerName, A-17
ServerPushLifespan, A-30

ServerPushMaxConnections, A-31
ServerPushMinTime, A-31 to A-32
ServerPushRefresh, A-29 to A-30
ServerRoot, A-16
StartServers, A-17
TimeOut, A-18
TransferLog, A-22 to A-23
TypesConfig, A-20
UserDNS, A-15
Server Resource Map Configuration, A-32 to A-45
AccessFileName, A-33
AddEncoding, A-37
AddIcon, A-41 to A-42
AddIconByEncoding, A-43
AddIconByType, A-42 to A-43
AddType, A-36
Alias, A-34 to A-35
DefaultIcon, A-39 to A-40
DefaultType, A-37 to A-38
DirectoryIndex, A-38
DocumentRoot, A-32 to A-33
ErrorDocument, A-44 to A-45
FancyIndexing, A-39
HeaderName, A-41
IndexIgnore, A-44
list of directives, 6-8
ReadmeName, A-40
Redirect, A-34
ScriptAlias, A-35 to A-36
wildcard expressions, A-1
Cookie VIs, 6-34 to 6-35
cookies
client-side cookies, 6-33
definition, 6-33
server-side cookies, 6-33
using cookies, 6-34 to 6-35
customer communication, *xiii*, C-1

D

data transfer. *See* FTP; FTP VIs.
DefaultIcon directive, A-39 to A-40
DefaultType directive, A-37 to A-38
Deny directive, A-9 to A-10
detailed mode, HTTP Server, 6-2 to 6-3
directives. *See* configuration directives.
Directory directive, A-2
DirectoryIndex directive, A-38
documentation
conventions used in manual, *xii*
organization of manual, *xi-xii*
related documentation, *xiii*
DocumentRoot directive, A-32 to A-33

E

electronic support services, C-1
e-mail capabilities, 1-4 to 1-5
e-mail support, C-2
E-mail VIs, 3-1 to 3-5
character sets, 3-2 to 3-4
definition, 3-3
ISO Latin-1, 3-3
Macintosh, 3-4
overview, 3-2 to 3-3
transliteration, 3-4
US-ASCII, 3-3
high-level, 3-2, 3-5
intermediate, 3-2
list of VIs, 3-5
low-level, 3-1
purpose and use, 3-1
environment variables for CGI applications (table), 6-20 to 6-21
error codes, B-1 to B-4
ErrorDocument directive, A-44 to A-45
ErrorLog directive, A-21 to A-22

F

FancyIndexing directive, A-39
File path format, in configuration files, 6-5
File Path VIs, 6-23 to 6-24
File Transfer Protocol. *See* FTP.
files and directories for Internet Developers Toolkit for G, 1-4
FTP
    active connection, 2-1
    anonymous, 1-6, 2-1
    ASCII mode, 2-2
    Image mode, 2-2
    passive connection, 2-1
    purpose and use, 2-1
FTP configuration, 1-6
FTP technical support, C-1
FTP VIs, 2-1 to 2-4
    high-level, 2-3
        list of VIs, 2-3 to 2-4
    intermediate, 2-2
    low-level, 2-2
    purpose and use, 2-1 to 2-2

G

G Web Server, 6-1 to 6-35
    CGI applications, 6-18 to 6-21
        application process (figure), 6-19
        CGI VI example, 6-21
        definition, 6-18
        environment variables (table), 6-20 to 6-21
        GET method of supplying parameters, 6-19 to 6-20
        POST method of supplying parameters, 6-19
        structure of CGI application (figure), 6-21
    CGI Utility VIs, 6-22 to 6-32
        categories, 6-22
        CGI Parameter VIs, 6-22 to 6-23
        File Path VIs, 6-23 to 6-24
        HTML VIs, 6-26 to 6-32
            HTML Document VI (example), 6-29
            HTML Generic Tag VI (example), 6-28
            HTML table block diagram, 6-27
            HTML+ Monitor VI (example), 6-28
            HTML+ String Array to Table VI (example), 6-27 to 6-28
            list of VIs, 6-30 to 6-32
        Keyed Array VIs, 6-25 to 6-26
        Network VIs, 6-24
        Panel Image VIs, 6-24 to 6-25
    configuration, 6-4 to 6-8
        Access Configuration, 6-5
            list of directives, 6-6
        required files, 6-5
        rules for configuration files, 6-4 to 6-5
        Server Configuration, 6-6 to 6-7
            list of directives, 6-6 to 6-7
        Server Resource Map Configuration, 6-7 to 6-8
            list of directives, 6-8
    definition, 6-1
    HTML, 6-8 to 6-10
    HTTP Server, 6-1 to 6-3
    maintaining client state information, 6-32 to 6-35
        client-side cookies, 6-33
        Cookie VIs, 6-34 to 6-35
        server-side cookies, 6-33
        using cookies, 6-34 to 6-35
    purpose and use, 6-3 to 6-4
    starting, 6-1
    URLs for panel images, 6-12 to 6-18
        animated panel image (.monitor URL), 6-14 to 6-16
        animated panel image (.spool URL), 6-16 to 6-18

panel image formats, 6-13
static panel image (.snap URL), 6-13 to 6-14
working with VIs, 6-10 to 6-12
embedding images, 6-12
executing VIs, 6-11 to 6-12
viewing running VIs, 6-10 to 6-11
GET method of supplying CGI parameters, 6-19 to 6-20
Global Access Configuration file, 6-5

H

HeaderName directive, A-41
high-level VIs
E-mail VIs, 3-2, 3-5
FTP VIs, 2-3 to 2-4
HTML, 6-8 to 6-10
definition, 6-8
standards, 6-9
tags, 6-8 to 6-9
HTML VIs, 6-26 to 6-32
HTML Document VI (example), 6-29
HTML Generic Tag VI (example), 6-28
HTML table block diagram, 6-27
HTML+ Monitor VI (example), 6-28
HTML+ String Array to Table VI (example), 6-27 to 6-28
list of VIs, 6-30 to 6-32
HTTP (HyperText Transfer Protocol) configuration, 1-7
HTTP Server, 6-1 to 6-3
detailed mode
Active CGIs, 6-3
Active Connections, 6-3
Cached CGIs, 6-3
definition, 6-1
example (figure), 6-2
Log information, 6-3
Pending Connections, 6-3
Server Activity information, 6-3
Server-Push Connections, 6-3
simple mode
definition, 6-1
example (figure), 6-1
STOP button, 6-2
HTTP Server Control VI, 6-1
HyperText Transfer Protocol (HTTP) configuration, 1-7
HyperText Markup Language (HTML). *See* HTML.

I

Image mode for FTP, 2-2
images
embedding, 6-12
panel. *See* panel images.
IndexIgnore directive, A-44
installation, 1-1 to 1-4
files and directories, 1-4
Macintosh, 1-2
UNIX, 1-3
Windows, 1-1 to 1-2
intermediate VIs
E-mail VIs, 3-2
FTP VIs, 2-2
Internet Developers Toolkit for G
components, 1-1
configuration, 1-6 to 1-7
[ftp] configuration, 1-6
[http] configuration, 1-7
[smtp] configuration, 1-6 to 1-7
installation, 1-1 to 1-4
files and directories, 1-4
Macintosh, 1-2
UNIX, 1-3
Windows, 1-1 to 1-2
purpose and use, 1-4 to 1-5
building CGI programs in G, 1-5
converting VIs to Internet applications, 1-5
e-mail and FTP features, 1-4 to 1-5

internet.ini file, 1-2, 1-4
ISO Latin-1 character set, 3-3
ISO-8859-1 character set, 3-3

J

JPEG (Joint Photographic Experts Group) format, 6-13

K

Keyed Array VIs, 6-25 to 6-26

L

Limit directive, A-7 to A-8
Log information, HTTP Server, 6-3
LogOptions directive, A-26 to A-27
low-level VIs
    E-mail VIs, 3-1
    FTP VIs, 2-2

M

Macintosh character set, 3-4
manual. *See* documentation.
.monitor URL, 6-14 to 6-16
    examples, 6-16
    purpose and use, 6-14 to 6-15
    syntax, 6-15 to 6-16

N

negotiated options, 4-2
Network Virtual Terminal (NVT), 4-1
Network VIs, 6-24
NVT (Network Virtual Terminal), 4-1

O

OnDeny directive, A-13 to A-14
Order directive, A-8 to A-9

P

Panel directive, A-2 to A-3
Panel Image VIs, 6-24 to 6-25
panel images
    animated panel image (.monitor URL), 6-14 to 6-16
    animated panel image (.spool URL), 6-16 to 6-18
    formats, 6-13
    static panel image (.snap URL), 6-13 to 6-14
PanelImageCacheCompactTime directive, A-29
PanelImageDepth directive, A-28
PanelImageQuality directive, A-28 to A-29
PanelImageType directive, A-27
passive connection for FTP, 2-1
Pending Connections, HTTP Server, 6-3
Per-Directory Access Configuration file, 6-5
PNG image format, 6-13
Port directive, A-14 to A-15
Portable Network Graphics (PNG) image format, 6-13
POST method of supplying CGI parameters, 6-19

R

ReadmeName directive, A-40
Redirect directive, A-34
RefererLog directive, A-24 to A-25
Require directive, A-11 to A-12
ResourceConfig directive, A-19

S

Satisfy directive, A-12 to A-13
ScriptAlias directive, A-35 to A-36
Server. *See* G Web Server; HTTP Server.
Server Activity, HTTP Server, 6-3
Server Configuration, 6-6 to 6-7
    list of directives, 6-6 to 6-7
    purpose, 6-6

Server Configuration directives, A-14 to A-32
AccessConfig, A-18 to A-19
AgentLog, A-23 to A-24
CGICacheTime, A-20 to A-21
ErrorLog, A-21 to A-22
list of directives, 6-6 to 6-7
LogOptions, A-26 to A-27
PanelImageCacheCompactTime, A-29
PanelImageDepth, A-28
PanelImageQuality, A-28 to A-29
PanelImageType, A-27
Port, A-14 to A-15
RefererLog, A-24 to A-25
ResourceConfig, A-19
ServerAdmin, A-15 to A-16
ServerName, A-17
ServerPushLifespan, A-30
ServerPushMaxConnections, A-31
ServerPushMinTime, A-31 to A-32
ServerPushRefresh, A-29 to A-30
ServerRoot, A-16
StartServers, A-17
TimeOut, A-18
TransferLog, A-22 to A-23
TypesConfig, A-20
UserDNS, A-15
Server Resource Map Configuration, 6-7 to 6-8
list of directives, 6-8
purpose, 6-7
Server Resource Map Configuration directives, A-32 to A-45
AccessFileName, A-33
AddEncoding, A-37
AddIcon, A-41 to A-42
AddIconByEncoding, A-43
AddIconByType, A-42 to A-43
AddType, A-36
Alias, A-34 to A-35
DefaultIcon, A-39 to A-40
DefaultType, A-37 to A-38
DirectoryIndex, A-38
DocumentRoot, A-32 to A-33
ErrorDocument, A-44 to A-45
FancyIndexing, A-39
HeaderName, A-41
IndexIgnore, A-44
list of directives, 6-8
ReadmeName, A-40
Redirect, A-34
ScriptAlias, A-35 to A-36
ServerAdmin directive, A-15 to A-16
ServerName directive, A-17
Server Push Connections, HTTP Server, 6-3
ServerPushLifespan directive, A-30
ServerPushMaxConnections directive, A-31
ServerPushMinTime directive, A-31 to A-32
ServerPushRefresh directive, A-29 to A-30
ServerRoot directive, A-16
server-side cookies, 6-33
Simple Mail Transfer Protocol (SMTP) configuration, 1-6 to 1-7
simple mode, HTTP Server, 6-1
SMTP (Simple Mail Transfer Protocol) configuration, 1-6 to 1-7
SMTP VIs. *See* E-mail VIs.
.snap URL, 6-13 to 6-14
examples, 6-14
purpose and use, 6-13
syntax, 6-13 to 6-14
spaces, in configuration files, 6-4
.spool URL, 6-16 to 6-18
examples, 6-17 to 6-18
purpose and use, 6-16
syntax, 6-17
starting the G Web Server, 6-1
StartServers directive, A-17
static panel image (.snap URL), 6-13 to 6-14
examples, 6-14
purpose and use, 6-13
syntax, 6-13 to 6-14

T

technical support, C-1
Telnet connection
definition, 4-1
negotiated options, 4-2
Network Virtual Terminal, 4-1
Telnet Protocol, 4-1
Telnet VIs, 4-2 to 4-3
list of VIs, 4-2 to 4-3
purpose and use, 4-2
TimeOut directive, A-18
TransferLog directive, A-22 to A-23
transliteration of character sets, 3-4
TypesConfig directive, A-20

U

Universal Resource Locators (URLs).
See URL VIs; URLs.
URL VIs, 5-2
list of VIs, 5-2
purpose and use, 5-2
URLs
CGI Parameter VIs, 6-22 to 6-23
definition, 5-1
examples, 5-1
for panel images, 6-12 to 6-18
animated panel image (.monitor URL), 6-14 to 6-16
animated panel image (.spool URL), 6-16 to 6-18
panel image formats, 6-13
static panel image (.snap URL), 6-13 to 6-14
US-ASCII character set, 3-3
UserDNS directive, A-15

V

VIs
CGI Utility VIs, 6-22 to 6-32
active, 6-3
cached, 6-3
categories, 6-22
CGI Parameter VIs, 6-22 to 6-23
File Path VIs, 6-23 to 6-24
HTML VIs, 6-26 to 6-32
Keyed Array VIs, 6-25 to 6-26
Network VIs, 6-24
Panel Image VIs, 6-24 to 6-25
converting to Internet applications, 1-3
Cookie VIs, 6-34 to 6-35
E-mail VIs, 3-1 to 3-5
high-level VIs, 3-2, 3-5
intermediate VIs, 3-2
low-level VIs, 3-1
purpose and use, 3-1
embedding images, 6-12
executing VIs, 6-11 to 6-12
File Path VIs, 6-23 to 6-24
FTP VIs, 2-1 to 2-4
high-level, 2-3 to 2-4
intermediate, 2-2
low-level, 2-2
purpose and use, 2-1 to 2-2
HTML VIs, 6-26 to 6-32
HTML Document VI (example), 6-29
HTML Generic Tag VI (example), 6-28
HTML table block diagram, 6-27
HTML+ Monitor VI (example), 6-28
HTML+ String Array to Table VI (example), 6-27 to 6-28
list of VIs, 6-30 to 6-32
HTTP Server Control VI, 6-1
Keyed Array VIs, 6-25 to 6-26
Network VIs, 6-24
Panel Image VIs, 6-24 to 6-25
Telnet VIs, 4-2 to 4-3
list of VIs, 4-2 to 4-3
purpose and use, 4-2
URL VIs, 5-2
list of VIs, 5-2
purpose and use, 5-2
viewing running VIs, 6-10 to 6-11

W

Web Server. *See* G Web Server.
white space, in configuration files, 6-4
wildcard expressions, A-1