

Customizing and Using the HTTP Server

This chapter covers the following topics:

- Initializing the HTTP Server
- Termination
- Operator Commands
- Configuration
- HTTP Server Parameters
- Content Processing
- Configurable Tables
- Default URL Processing
- Resource Usage
- Pooled Server Processing
- Conversational Processing

Initializing the HTTP Server

The HTTP server is normally initialized by specifying a `SERVER` statement:

```
SERVER=(HTTP,HAENSERV,Configuration=<config>)
```

-where

HTTP	is the name of the <code>SERVER</code> . The name of the server may be any name valid for the <code>SERVER</code> statement.
HAENSERV	is the name of the HTTP server module. The HTTP server module must always be specified as <code>HAENSERV</code> .
<config>	is the name of the HTTP server configuration to be used. The configuration statement may specify any valid load module name. The load module must have been generated as described in this section; otherwise, the results are unpredictable. The default configuration name used is <code>HAANCONF</code> .

If inactive for any reason, the HTTP server may also be started using the operator command

```
SERV,INIT,HTTP,HAENSERV,Configuration=<config>
```

-where the meaning of the parameters is the same as previously outlined for the `SERVER` statement.

Notes:

1. 1. It is possible to have more than one HTTP server active within the same SMARTS server address space at the same time. Each server requires a different server name and a different configuration to ensure that it does not attempt to use the same port as one of the other servers.
2. 2. The SMARTS environment must be active before any the HTTP server can be activated; otherwise, the HTTP server initialization fails. This can be achieved by placing any of the HTTP server SERVER statements after the SERVER statement for the SMARTS environment in the SMARTS server sysparms input file. When the servers are started using operator commands, the SMARTS environment must be started first.
3. . The QUIESCE command should normally be issued first to give the server time to stop accepting new requests and finish processing any old requests.
4. . The first time the command is issued, the listening program attached when the server was initialized is canceled if it is still active. In this case, the terminate must be requested again as the HTTP server cannot terminate properly until its associated listening task has terminated.
5. . If the SMARTS environment is terminated, all the HTTP server listening tasks are automatically canceled.
6. . It is not possible to terminate the SMARTS environment unless all the HTTP servers are first terminated.

Termination

The HTTP server is terminated automatically when the SMARTS server address space is terminated. However, it is also possible to cycle the server without bringing the address space down by issuing the operator commands

```
SERV,HTTP,QUIESCE
SERV,TERM,HTTP
```

-where HTTP is the name given to the server at startup.

Notes:

1. . The QUIESCE command should normally be issued first to give the server time to stop accepting new requests and finish processing any old requests.
2. . The first time the command is issued, the listening program attached when the server was initialized is canceled if it is still active. In this case, the terminate must be requested again as the HTTP server cannot terminate properly until its associated listening task has terminated.
3. . If the SMARTS environment is terminated, all the HTTP server listening tasks are automatically canceled.
4. . It is not possible to terminate the SMARTS environment unless all the HTTP servers are first terminated.

Operator Commands

In addition to the SMARTS server commands to initialize and terminate the HTTP server, the following operator commands may be issued to the HTTP server by issuing the SMARTS server operator command

`SERV,HTTP,<command>`

-where

HTTP	is the name with which the HTTP server was started.
<command>	is one of the commands in the following table:

Command	Function
CLEARPOOL	Terminates all active HTTP pooled servers. If no pooled servers are in operation, this command has no effect.
QUIESCE	Causes the HTTP server to stop accepting new requests while enabling existing requests to continue normally. Software AG recommends issuing this command to the HTTP server before attempting to terminate it.
TERM	Terminates the HTTP server as gracefully as possible.
FORCE	Forcibly terminates the HTTP server. This command should only be used in emergencies as it may cause integrity problems.

Configuration

The HTTP server is configured by building a load module with the HMANCONF macro delivered with SMARTS.

Under the SMARTS server, the load module is identified on the SERVER statement or on the operator command used to start the server.

The load module is specified using the 'Configuration=' parameter and defaults to HAANCONF.

Any attempt to use a module that was not generated according to the instructions in this section will cause unpredictable results.

Sample HAANCONF Member

The HTPvrs.USER.SRCE statement contains a sample HAANCONF member. This may be copied and/or modified to produce a number of different configuration options, if required. A configuration option can then be selected by an operator when SMARTS is started.

Assembling the Configuration Member

Once the configuration source member has been created or modified, it must be assembled to produce the associated load module for use by the system.

It must be linked into a load library in the COMPLIB concatenation of the SMARTS server environment start-up procedure.

Software AG recommends that the HTPvrs.USER.LOAD dataset contain all configuration modules.

The member HJBNACNF on the HTPvrs.USER.SRCE dataset contains sample JCL to compile and link the delivered HAANCONF member.

HTTP Server Parameters

The following parameters may be specified on the HMANCONF macro:

CGIPARM

Parameter	Use	Possible Values	Default
CGIPARM	A parameter string to be passed to every CGI program that is started.	1-256 byte parameter string	none

The string is passed in standard OS/390 or MVS/ESA format where register 1 points to a pointer that points to a half word length followed by the data.

CGIPARM is used to pass parameters or set runtime options for language environment-enabled programs.

CONTBUFL

Parameter	Use	Possible Values	Default
CONTBUFL	The size of a buffer allocated by the HTTP request processing program to hold any HTTP content submitted with an HTTP request.	1-32000	1024

If the content size exceeds the value set in this parameter, the request is rejected.

Changing this size affects the amount of local storage that the request processing program needs to run.

CONV

Parameter	Use	Possible Values	Default
CONV	Indicates whether the server supports conversational users.	NO YES	NO

If NO, any program request to establish a conversation is rejected with an appropriate return and reason code.

DEFACEE

Note:

This parameter is only used in a SMARTS server environment.

Parameter	Use	Possible Values	Default
DEFACEE	Indicates whether the server will build a default ACEE for the user specified in HTTPUSER during initialization and startup.	NO YES	NO

Value	Description
YES	A default ACEE is built and associated with any user that does not log on to the system.
NO	Any user that does not log on to the system has the authority associated with the address space.

DFLTCONT

Parameter	Use	Possible Values	Default
DFLTCONT	The default content type to be assigned to a URL if the content type cannot be determined through the dataset name or member type processing mechanism.	HTTP content type header	APPLICATION/OCTET-STREAM

The web browser uses the content type header to determine what to do with data. For example, for content type TEXT/HTML, the browser interprets the data as an HTML page. For content type IMAGE/GIF, the browser attempts to interpret the data as a GIF file.

The default value causes the web browser to download as binary data any URLs for which the content cannot be explicitly determined; that is, such URLs are downloaded as is without any translation.

DFLTURL

Parameter	Use	Possible Values	Default
DFLTURL	The default URL to be returned to a user request connecting to the HTTP server with no URL information.	default URL for your installation	none

This circumstance occurs when the following URL is requested from a browser:

`http://your.ip.address:port`

Refer to the subsection later in this section relating to the specification of a default URL.

HTTPUSER

Note:

This parameter is only used in a SMARTS server environment.

Parameter	Use	Possible Values	Default
HTTPUSER	The user ID with which each HTTP request is identified if the HTTP request does not contain an authorization header.	1-8 character user ID	HTTPUSER

This is effectively the default user ID assigned initially to all requests.

HTTPLIST

Note:

This parameter is only used in a SMARTS server environment.

Parameter	Use	Possible Values	Default
HTTPLIST	The user ID assigned to the HTTP task that listens for requests.	1-8 character user ID	HTTPLIST

HTTPHCD

Note:

This parameter is only used in a SMARTS server environment.

Parameter	Use	Possible Values	Default
HTTPHCD	The hardcopy device name associated with any HTTP users.	1-8 character name	HTTPHCD

This output device name is used when the program attempts to write output to a SYSOUT/SYSLST type device. For example, when LE/370 is active and an abend occurs, LE/370 writes a dump to this hardcopy device name.

Output written to this device may be viewed and/or deleted using USPOOL or printed.

LOGON

Note:

This parameter is only used in a SMARTS server environment.

Parameter	Use	Possible Values	Default
LOGON	Determines the level of security that the HTTP server will enforce.	ALLOWED DISALLOWED REQUIRED	ALLOWED

The parameter must be used in association with the SMARTS server SECSYS configuration parameter.

Refer to the chapter on Security for more information about this parameter.

MSGCASE

Parameter	Use	Possible Values	Default
MSGCASE	The case in which messages are to be issued.	MIXED UPPER	MIXED

NATLIB

Parameter	Use	Possible Values	Default
NATLIB	Name of the default library where Natural CGI requests to the HTTP server are directed.	1-8 character name	NATCGI

If no library is provided on the CGI request, the program specified on the Natural CGI request to this server must be cataloged in this library.

NATPARAM

Parameter	Use	Possible Values	Default
NATPARAM	Parameter string to be provided as override parameters to Natural for all Natural CGI requests.	1-256 byte parameter string	none

Note:

The contents of this field are not validated in any way and may cause problems if invalid. Any string should be thoroughly tested before being set in this parameter.

The 'STACK' override parameter is ignored if specified in this string.

NATTHRD

Parameter	Use	Possible Values	Default
NATTHRD	Under the SMARTS server environment, the name of the thread-resident portion of Natural as created for the SMARTS server environment TP monitor.	1-8 character name	NATCOM

See Installing Natural CGI.

PORT

Parameter	Use	Possible Values	Default
PORT	The sockets port on which the HTTP server server should listen for incoming HTTP requests.	1-32000	80

RECVBUFL

Parameter	Use	Possible Values	Default
RECVBUFL	The length of the buffer used for receiving input data from the network.	1-32000	4096

When conversations are being supported, this buffer must be set to the highest incoming data size expected for one incoming conversational HTTP request; otherwise, data may be lost.

When conversations are not being supported, this buffer is reused to read the entire incoming data stream.

Changing this size affects the amount of local storage the request processing program needs to run.

SEND

Parameter	Use	Possible Values	Default
SEND	Determines how HTTP sends data in response to a request.	IMMEDIATE BUFFERED	IMMEDIATE

Value	Description
IMMEDIATE	Sends data as soon as it is available. Although it is more resource intensive, this option is useful where requests are failing: the web user receives whatever data has been created to the point of the failure.
BUFFERED	Buffers all data in the buffer created by the SENDBUFL parameter. If failures occur, the end user may see no response data at all as the data is being buffered; however, this is less resource and Entire Net-Work intensive. Because the BUFFERED setting greatly increases the performance of the server, Software AG recommends that use it in a production environment.

Note:

The SENDBUFL parameter must still be specified with SEND=IMMEDIATE as the data must be copied and translated in some instances prior to being sent.

SENDBUFL

Parameter	Use	Possible Values	Default
SENDBUFL	The length of the buffer used for sending output data to the network.	1-32000	4096

Changing this size affects the amount of local storage the request processing program needs in order to run.

SERVNAME

Parameter	Use	Possible Values	Default
SERVNAME	Name to identify the system.	1-8 character name	none

This name is included in all messages (except some start-up and termination messages) issued to the operator during the execution of the HTTP server.

The name may be used in the future by the HTTP server system to uniquely identify itself within a machine.

SERVPOOL

Parameter	Use	Possible Values	Default
SERVPOOL	Indicates whether the HTTP server maintains a pool of previously started HTTP servers.	NO YES	NO

Value	Description
NO	Pooled servers are not maintained and a new server is started for each HTTP request.
YES	The HTTP server reuses previously started servers, which can significantly enhance performance.

TRACE

Parameter	Use	Possible Values	Default
TRACE	Turns on tracing to the DD statement identified by the TRACEDD keyword.	HEADER DATA	none

One or both options may be specified. The options must be specified in parentheses. For example, the following turns both traces on:

```
TRACE=(HEADER,DATA)
```

Value	Description
HEADER	Dumps all HTTP headers and their associated data to the DD once they have all been read and processed.
DATA	Traces all HTTP input data as it is read and all data being sent in response to the request prior to it being sent. The data is printed in both character and hex formats. The hex represents what is actually sent in ASCII. The character output is translated to EBCDIC so that it can be read.

TRACEDD

Parameter	Use	Possible Values	Default
TRACEDD	Name of the DD to which all HTTP trace data is written.	1-8 character name	HTPTRCE

If the DD name is not specified in the SMARTS start-up procedure, it is automatically allocated as a SYSOUT dataset.

URLPBUFL

Parameter	Use	Possible Values	Default
URLPBUFL	Length of the buffer used for holding parameters passed on any given URL request (that is, any data following ? in the URL).	1-32000	256

Depending on the nature of the requests in use, the length could be increased or decreased; however, if there is insufficient space in this buffer, the request is rejected.

Changing this size affects the amount of local storage the request processing program needs in order to run.

Content Processing

One of the most important pieces of information that the HTTP server provides to the web browser in response to a HTTP request is the 'content type'. This is the HTTP header that the browser uses to interpret the data sent in response to the request. The HTTP server has a number of ways to determine what the content type is.

Member Type Processing

When a member type is specified on a URL request, even though the mainframe operating system has no concept of types as such, the HTTP server uses this type to look up a user configuration table HAANTYPT to determine whether it can identify the content type of the URL.

The source for the table HAANTYPT is delivered in HTPvrs.SRCE and copied during installation to the HTPvrs.USER.SRCE datasets for modification by the user. The table is built using the HMANTYPT macro which has two parameters:

TYPE	1-8 character member type.
CONTENT	The content type to be returned to the browser when a member of this type is requested on a URL. The HTTP server does not validate content types: any type may be specified and new types are constantly being created. The sample HAANTYPT member contains examples of commonly used content types.

Note:

The values specified for the above parameters are case-sensitive; for example, HTML and html are different 'TYPE's.

Once modified, the HAANTYPT table must be reassembled into a load library in the SMARTS server environment COMPLIB concatenation.

Software AG recommends that you use the HTPvrs.USER.LOAD dataset for this. Member HJBNTYPT in the HTPvrs.USER.SRCE dataset contains sample JCL to assemble and link the HAANTYPT table.

If a match cannot be found for the type specified on the URL, or no type is specified on the URL, the HTTP server attempts to determine the content type from the last level of the URL provided.

Dataset Name Processing

The HTTP server breaks down the URL provided by the user into

- a dataset name component; and
- optionally a member and member type component

```
/this/is/a/pds/member.type is THIS.IS.A.PDS(MEMBER)
```

-or

```
/this/is/a/seq/file/ is THIS.IS.A.SEQFILE
```

On OS/390 or MVS/ESA systems, the HTTP server assumes that the last level of the resulting dataset name indicates the sort of data contained in a given dataset. In the above examples, 'PDS' and 'FILE' are the last levels.

Once the last level is determined, the HTTP server checks a second user-configurable table HAANDSNT to determine if the last level can be used to map the URL to a content type. The HAANDSNT member is provided on the HTPvrs.SRCE dataset and copied by the installation to the HTPvrs.USER.SRCE for modification. HAANDSNT is built using the HMANDSNT macro which has the following parameters:

LASTDSNL	The last level of the dataset name resulting from the interpretation of the URL.
CONTENT	The content type to be returned to the browser when a member from this dataset (or the dataset itself, if sequential) is requested on a URL. The HTTP server does not validate these content types: any type may be specified and new types are being created daily. The sample HAANDSNT member contains examples of commonly used content types.

Note:

The values specified for the above parameters are case-sensitive; for example, HTML and html are different 'LASTLEVL's. At present, all LASTLEVL specifications should be in uppercase as there is no support for lowercase dataset names.

Once modified, the HAANDSNT table must be reassembled into a load library in the SMARTS server COMPLIB concatenation.

Software AG recommends that you use the HTPvrs.USER.LOAD dataset for this. Member HJBNDNT in the HTPvrs.USER.SRCE dataset contains sample JCL to assemble and link the HAANDSNT table.

If a match cannot be found in the HAANDSNT table, the HTTP server uses the default as specified by the DFLTCONT configuration parameter.

CGI Request Output Processing

Any CGI program, whether it is written in Natural, C, COBOL, or PL/1, may write a CONTENT-TYPE header to the output stream to indicate the content it is going to send. If the HTTP server detects that no CONTENT-TYPE header is sent by the CGI program, it sends a CONTENT-TYPE header using the default content type for the system as specified by the DFLTCONT configuration parameter.

Configurable Tables

A number of translation tables are supplied with the HTTP server in source format.

Do not modify these tables, as incorrect modification may cause server problems for the server.

If you find errors in these tables, contact Software AG so that corrections can be made generally available.

HAANEUTT

The HAANEUTT table translates from EBCDIC to uppercase EBCDIC.

HAANIPTT

The HAANIPTT table translates from ASCII to EBCDIC.

HAANIUTT

The HAANIUTT table translates from ASCII to uppercase EBCDIC.

HAANOPTT

The HAANOPTT table translates from EBCDIC to ASCII.

HAANTOTT

The HAANTOTT table translates trace character output to ensure valid output data.

Default URL Processing

Care must be taken with the default URL specified for the HTTP server. If a file containing an HTML page is set as the default, this file must not contain any relative URLs as any attempt to reference these relative URLs from a page accessed as the default URL will fail.

The reason for this is that the browser sends a request for the dataset named `'/'` which causes the HTTP server to use the default URL specified in the configuration parameters. If this is a dataset, it will have the form

```
/a/b/c/member.type
```

-and `'member'` will be returned to the caller.

Any relative URLs in that member are relative to `/a/b/c/`; however, the browser does not know this and assumes that they are relative to `'/'`, which is how it originally found it.

The browser translates the URL `'./graphics/my.gif'` to be `'/graphics/my.gif'` when, in fact, the URL should be `'/a/b/c/graphics/my.gif'`.

As the HTTP server knows nothing about a dataset called simply `'graphics'` with a member of `'my'`, the request fails.

There are two ways to avoid this problem:

- In the default URL, specify absolute URLs only. This must only be done for one HTML file and as such should not be a major problem to maintain.
- Specify a CGI program as the default URL which either directs the web user to the correct URL or rejects the request depending on how secure the server should be.

Resource Usage

The HTTP server itself uses a combination of local storage and global storage acquired explicitly by direct requests for the storage and implicitly by using the SMARTS API requests.

The following sections outline the use of HTTP server storage.

Global Storage

The following storage areas used by the HTTP server are allocated in global storage above the 16 megabyte line in all environments:

Storage Area	Size in Bytes
HTTP main control block (HMCB)	2048
Server storage stack	4096
Storage for pool server processing	<no of pooled servers>*64
Storage for conversational users	<no of conversations>*64
Storage for data pipe between processes. This is relatively short term as it is allocated, written, read, and freed.	<max of RECVBUFL> *<concurrent pipes>

HAANLIST Storage

The HAANLIST program listens on the specified port for HTTP requests and is fully reentrant. The following storage areas are allocated for its processing needs:

Storage Area	Size in Bytes
Working storage	256
Storage required for 2 sockets	see the SMARTS environment estimates
Various working storage areas	1024

HAANRQST Storage

This program actually processes the HTTP request and is fully reentrant. The following storage areas are allocated for its processing needs:

Storage Area	Size in Bytes
Working storage	256
Storage required for 1 socket	see the SMARTS environment estimates
Various working storage areas	1024
Storage for the HTTP request processing block	4096
Storage for the receive buffer	RECVBUFL specification
Storage for send buffer	SENDBUFL specification
URL parameter buffer	URLPBUFL specification
Content buffer length	CONTBUFL specification
For each request header received:	32+1+request header name and header value

Additional Storage Used for CGI Requests

When a CGI request is processed, all of the headers must be set up as environment variables for the CGI program; therefore, the following additional storage is allocated:

Storage Area	Size in Bytes
Environment variable overhead	see the SMARTS environment estimates
Additional data per variable	HTTP request header length + data length + 1

Com-plete Considerations

Any additional storage required by the CGI program must be considered when calculating the thread size and ULIB catalog size required by a given CGI program (in this case, the PAENSTRT program) to function.

Pooled Server Processing

The HTTP server implements pooled server processing dynamically.

When a request is received, the HTTP server checks the pooled server queue to determine if any pool servers are available. For the first request, no pool server is found so the system starts a server to process the request. When the request is processed, the server puts itself on the pooled server processing queue. When the next request arrives, the pooled server is found on the queue and is used to process the request.

Advantages of Pooled Servers

- Using a pooled server rather than starting a new server saves the cost of starting and initializing a new thread of control and terminating the request processing logic. Significant resources are thus saved.

- The system creates pooled servers until a steady state is reached where a pooled server is always available to process an incoming request.
- If a pool server ABENDs for some reason, only the user being processed is affected. If insufficient pooled servers are available, the HTTP server attaches more users.
- Pooled servers are dormant while on the pooled server queue. They use no CPU. In Complete environments, pooled servers are eligible for rollout so they do not occupy a thread.
- The CLEARPOOL operator command makes it possible to clear the pooled server queue and terminate all active pooled servers.

Considerations when Using Pooled Servers

A pooled server can process any type of request. For example, the first request it services may be a static HTML deliver request, the second may be a COBOL program CGI request, the third may be a Natural program CGI request.

- The amount of storage available in the pooled server's thread must be carefully considered to ensure that sufficient unfragmented storage is available to run whatever request may arrive. To that end, it may be advisable to have a separate HTTP server processing Natural requests only.
- Because a pooled server instance never really terminates, CGI programs must clean up after processing to avoid a negative impact on following requests.

Natural Considerations

When a Natural CGI request is issued and a new thread must be started, the Natural interface is started and, using an internal inverted call mechanism, Natural CGI programs are driven.

Using the inverted mechanism to call Natural means that the Natural environment is only started once per pooled server so that the next time a Natural CGI request is handled by the same pooled server, no Natural initialization/termination is required.

In this way, a significant savings is realized in both CPU and wall clock time required to process Natural CGI requests.

Conversational Processing

Instead of maintaining a generic queue of active user threads, a more specific queue is maintained for conversational processing.

A conversation is maintained by sending a session-specific cookie to the browser with the output from the CGI program. The cookie is returned with the next request that enables the HTTP server to match the incoming request with the active request on the conversational request queue.

The incoming data is piped to the program in conversation and the conversational program is dispatched again to process the request from the user with whom it is in conversation.

If a conversational program terminates due to a timeout or an ABEND, the next time the user attempts to converse with the other user in the conversation, a message is issued to indicate that the conversation no longer exists.

