

# Running CGI Programs under SMARTS

This chapter provides information about running application programs under SMARTS in the supported environments.

This chapter covers the following topics:

- The SMARTS Server Environment
  - The Complete Environment
  - Natural Considerations
  - C Considerations
  - COBOL Considerations
  - PL/1 Considerations
  - S/390 Assembler Considerations
- 

## The SMARTS Server Environment

Programs that comply with the HTTP server CGI requirements can use any of the mechanisms available under the SMARTS server environment to access data.

It is currently possible to access ADABAS, DB2, and VSAM data from this environment.

Refer to the appropriate SMARTS documentation for more information about accessing data from this environment.

This section provides information about running programs in the SMARTS server environment.

### Linking the Program

An application program may only be run in the SMARTS server environment if it is available to the SMARTS server address space. To make an application program available to the SMARTS server address space, link the program into a dataset in the COMPLIB concatenation under MVS or MSP, or to one of the libraries in the VSE search chain. Always link a reentrant module with the 'RENT' option.

Sample jobs HJBNC0BC and HJBNPL1C are provided on the HTPvrs.SRCE dataset to do this once an object module has been created. Creation of the object modules is covered in the appropriate language-specific section of this chapter.

**Note:**

Since high-level languages like Natural, COBOL, and PL/1 load the SMARTS-provided CGI extensions, COBOL and PL/1 programs must be linked into a dataset in the COMPLIB concatenation whereas no additional processing is required for Natural.

## Requirement

All CGI programs must run RMODE=ANY.

## The Com-plete Environment

Programs that comply with the HTTP server CGI requirements can use any of the mechanisms available under Com-plete to access data.

It is currently possible to access ADABAS, DB2, and VSAM data from this environment.

Refer to the appropriate Com-plete documentation for more information about accessing data from this environment.

This section provides information about running programs under Com-plete.

### Linking the Program for Com-plete

An application program may only be run under Com-plete if it is available to the Com-plete address space. To make an application program available to the Com-plete address space, link the program into a dataset in the COMPLIB concatenation under MVS or MSP, or to one of the libraries in the VSE search chain. Always link a reentrant module with the 'RENT' option.

Sample jobs HJBNCOBC and HJBNPL1C are provided on the HTPvrs.JOBS dataset to do this once an object module has been created. Creation of the object modules is covered in the appropriate language-specific section of this chapter.

#### Note:

Since high-level languages like Natural, COBOL, and PL/1 load the SMARTS-provided CGI extensions, COBOL and PL/1 programs must be linked into a dataset in the COMPLIB concatenation whereas no additional processing is required for Natural.

### Preparing Com-plete for the Application

In general, the user must tell Com-plete about the load module before it will execute successfully in these environments. This process is called 'cataloging' the program and is achieved using the Com-plete ULIB utility. The ULIB utility is provided with the size of the thread below the line in which the program will run. If this calculation is incorrect, the application program is unlikely to run correctly. This is discussed in the next section. Other program-related options may be specified to the ULIB utility as well. Refer to the Com-plete Utilities Manual for more information.

### Calculating the Catalog Size under Com-plete

All applications running under Com-plete have two areas of storage available to them: thread storage above and below the 16-megabyte line.

- Storage above the line is the same for all threads and is available in its entirety to any program running in a given thread, regardless of the catalog size specified to the ULIB utility for the program.

- Storage below the line is limited and therefore more tightly controlled. The amount available to an application is controlled by the storage specification set for the program using the ULIB utility.

**Note:**

The maximum size any given thread can handle is the thread size specified in the Com-plete configuration parameters minus 4k. Thus 496K is the largest application program catalog size that a 500K thread can handle.

The catalog size comprises all the storage the program allocates locally as follows:

- The size of the program if it is loaded into the thread.

The program is loaded into the thread if it is not specified as a Com-plete RESIDENTPAGE program.

- Any working storage allocated by the program.

For languages such as C, C++, COBOL, and PL/1, a table is generally built that indicates the initial values of storage to be allocated. Software AG recommends that you use the sample table provided for batch in each case (only the language environment case if all programs are LE-enabled) and modify it to reflect the storage requirements of the online programs that will run under Com-plete. The language-related documentation or the language environment (LE) documentation tells you how to calculate the storage requirement for any given program.

- Any storage allocated from the thread by the system.

These storage estimates are found in the section Resource Usage.

- Storage for any other programs (not in the RESIDENTPAGE area) called by the main application program and the storage they require from the thread area.

When calculating these values, add 5 to 10% for fragmentation of the storage in the thread area.

## Catalog Size for CGI Programs under Com-plete

Because a program that runs as a CGI program is not the first program loaded into the thread, its ULIB definitions are not used to build the thread.

Instead, the ULIB definitions for the PAENSTRT module are used. PAENSTRT must be cataloged large enough to accommodate itself and the CGI programs that it calls.

The storage used by the HTTP server request processing module is documented in the section Resource Usage , while the storage used by the CGI program may be calculated as documented in the previous section.

## Program Index Entries under Com-plete

To avoid repeating the same load process every time, Com-plete keeps an index of the most used program names in core. This index maintains information about the program in the load library so that the process of loading the program is quicker.

If the program is relinked, it is generally not found automatically because the index still has a record of the older version of the program. The ULIB utility is used to inform Com-plete that the information about the module must be refreshed.

Com-plete also remembers when a module was not found. If a module was not found and is subsequently added to the load library, you must inform Com-plete that it now exists by refreshing the module in the same way.

## Running the Program under Com-plete

After you have logged on through the VTAM interface, you may run the program from the command line of the USTACK screen. Alternately, you may invoke the program by issuing a CGI request for the program to an HTTP server running in the Com-plete environment where the program is available.

## Program Options or Functions to Avoid under Com-plete

Compilers offer a number of facilities and options to control the execution of a program. In general, options or functions that impact the following areas should be avoided:

- Abnormal termination recovery or diagnostics processing can impact the data in any subsequent dump and render problem resolution even more difficult. Any dumps or diagnostics provided for support purposes must be generated with these options turned off.
- Use options that load the SMARTS extension modules dynamically at runtime. Otherwise, these modules must be linked to the application program modules, which may cause problems with version control and also makes the module larger. When the language's runtime system loads these modules, they are found in the RESIDENTPAGE area and shared among all application programs and languages.
- Use the SMARTS stdio functions to access sequential operating system files when required. Using I/O functions to access this data will either fail or impact the integrity of the Com-plete system.

## Recommendations for the Com-plete Environment

Refer to the Complete System Programmer's Manual for information about using LE/370 with Com-plete.

Software AG recommends that you define the following programs as RESIDENTPAGE:

```
RESIDENTPAGE=CEEBINIT
RESIDENTPAGE=CEEEV005
RESIDENTPAGE=CEEPLPKA
RESIDENTPAGE=IGZCFCC
RESIDENTPAGE=IGZCLNK
RESIDENTPAGE=IGZCPAC
RESIDENTPAGE=IGZCPCO
RESIDENTPAGE=IGZCULE
RESIDENTPAGE=IGZCXFR
RESIDENTPAGE=IGZEINI
RESIDENTPAGE=IGZETRM
RESIDENTPAGE=IGZEVEX
```

## Recommendations for Cobol Running under Com-plete

Software AG recommends that you use the following parameters:

```
STACK=( 8K, 8K, BELOW, KEEP ),
STORAGE=( 00, NONE, 00, 8K ),
```

Software AG also recommends that you assemble and link CEEUOPT to the COBOL program.

## Natural Considerations

Natural must be installed in the SMARTS server environment before SMARTS functions can be used.

### Running Natural Applications

Natural applications that use SMARTS functions are run as normal applications; however, the SMARTS environment must be active and, to run Natural CGI requests, the HTTP server must be active.

### Natural and the SMARTS CGI Extensions

#### To use the SMARTS CGI extensions with Natural

1. INPL it into an appropriate Natural environment.
2. include the local data area HNANCGRL in the HTPvrs Natural library (provided as part of the SMARTS installation materials); and

The HNANCGRL LDA defines the status area and codes that may be returned to the various SMARTS CGI extensions interface requests.

Once HNANCGRL has been included, the Natural program may simply issue calls as follows:

```
CALL 'HAANUPR' HTPCGI-status function parm1 parm2 < .. > parmn
```

The following calls may also be issued but are only included for compatibility with previous releases. The HAANUPR interface module implements the recommended interface: the following interface requests will not be enhanced:

```
CALL 'HAANCGIG' HTPCGI-status field value length
name1 value-length1 value1 type1
name2 value-length2 value2 type2
< .. >
namen value-lengthn valuen typen
CALL 'HAANCGIP' HTPCGI-status data length
CALL 'HAANCGIT' HTPCGI-status data length
```

The parameters required by these interfaces are described in the section SMARTS CGI Extensions . The Natural format required for the other parameters is as follows:

Name	Natural Format	Description
field	A<length>	Must be an alpha field ending in a blank character
variable	A<length>	Must be an alpha field ending in a blank character
value	A<length>	Must be an alpha field of length 'length'
length	I2	Must be a two-byte binary field containing a length
data	no defined format	The start of an area (normally alpha) of length 'length'

## Natural Script

Using SMARTS and the Natural ISPF macro facility, you have the option to generate dynamic HTML using the Natural language. The process is known as macro expansion, where the text (or HTML in this case) is generated. This can also consist of variable substitution, repeating blocks, conditionally generated text, along with file I/Os.

The macro facility is an extension of the Natural language and comprises two types of statements: processing statements and text lines. Both are identified by the macro character that is defined by the Natural ISPF administrator.

### Processing Statements

Processing statements are executed during the macro expansion. The statements must be preceded by the macro character and followed by a blank. The full power of the Natural language is available for processing statements.

### Text Lines

Text lines are copied to the generated output of the macro. They can contain variables or text. A variable must be preceded by the macro character in order to be interpreted during expansion.

### Example

In the following example, the macro character is ^. Lines 0010, 0020, and 0040 are processing statements; line 0030 is a text line. Note that the ^ character is in front of the #NAME text line variable without a space between.

Example:

```
0010 ^ MOVE 'DAVE' TO #NAME
0020 ^ IF #NAME EQ 'DAVE' THEN
0030 <b> ^#NAME </b>
0040 ^ END-IF
```

When the example above is expanded, the generated output is as follows:

```
<b> DAVE </b>
```

## How It All Works with SMARTS

Although its use is not required, the Natural ISPF macro facility provides a way to simplify the programming of dynamic HTML generation.

Using Natural CGI, you can execute Natural objects directly from the browser. Within the Natural object, calls can be included to get variables from the HTML page the call was initiated from and also to put output back to the browser.

To start, an object of type macro must be created. This object type is only available within Natural ISPF. After the macro is created, Natural code can be added for dynamic generation. The macro objects reside in Natural libraries. When using Natural CGI, all objects must reside in the NATCGI library. Once a Natural ISPF macro is stowed, it can be executed outside of Natural ISPF. When a macro is executed within Natural ISPF, the output is written to the workpool (a Natural ISPF facility). When a macro object is executed in native Natural, the output is written to the editor source area.

To put it all together, the HTML and Natural code are added to the macro object. Once the object has been tested, it is stowed. Within the macro is a call to the HANCGIP program that uses a Natural routine to read the editor source area and write the lines back to the browser.

### Macro Object Example:

```

0010 ^ DEFINE DATA LOCAL
0020 ^ 01 #TITLE (A10)
0030 ^ END-DEFINE
0040 ^ ASSIGN #TITLE = 'This is a Natural Script test'
0050 <HTML>
0060 <HEAD>
0070 <TITLE> ^#TITLE </TITLE>
0080 <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=iso-8859-1">
0090 <META NAME="AUTHOR" CONTENT="A.N. Other">
0100 <META NAME="FORMATTER" CONTENT="Microsoft FrontPage 2.0">
0110 <META NAME="GENERATOR" CONTENT="Mozilla/3.01Gold (WinNT; I) [Netscape]">
0120 </HEAD>
0130 <BODY TEXT="#000000" BGCOLOR="#FFFFFF" LINK="#0000EE" VLINK="#551A8B"
0140 ALINK="#FF0000">
0150
0160 <P><IMG SRC="../images/abc.gif" HEIGHT=28 WIDTH=378><BR>
0170 <A HREF="/imagemap/barmenu.map"><IMG ISMAP SRC="../images/BARMENU.gif"
0180 BORDER=0 HEIGHT=28 WIDTH=378></A></P>
0190
0200 <H2>This is a test </H2>
0210 </BODY>
0220 </HTML>
0230 ^ FETCH RETURN 'HANCGIP'
```

The only Natural lines in this example are on 0010, 0020, 0030, 0040, and 0230. These are all processing statements. The rest of the lines are all text lines. Note that the variable #TITLE is preceded by the macro character on line 0070. During the macro expansion, this variable is substituted with the actual value of the variable.

The line 0230 is calling a Natural program called HANCGIP. As mentioned earlier, the output is written to the editor source area when macros are executed outside of Natural ISPF. HANCGIP reads all lines in the editor source area and puts them back to the browser. It is that easy to create dynamic HTML. When you want to change the HTML, you can create new HTML using a tool such as FrontPage and then just cut and paste it into the macro. Then insert the Natural code for dynamic generation, stow the object, and

test it.

After the macro is expanded, the output written to the editor source area looks like the following:

```
0010 <HTML>
0020 <HEAD>
0030 <TITLE> This is a Natural Script test </TITLE>
0040 <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=iso-8859-1">
0050 <META NAME="AUTHOR" CONTENT="A.N. Other">
0060 <META NAME="FORMATTER" CONTENT="Microsoft FrontPage 2.0">
0070 <META NAME="GENERATOR" CONTENT="Mozilla/3.01Gold (WinNT; I) [Netscape]">
0080 </HEAD>
0090 <BODY TEXT="#000000" BGCOLOR="#FFFFFF" LINK="#0000EE" VLIINK="#551A8B"
0100 ALINK="#FF0000">
0110
0120 <P><IMG SRC="../images/abc.gif" HEIGHT=28 WIDTH=378><BR>
0130 <A HREF="/imagemap/barmenu.map"><IMG ISMAP SRC="../images/BARMENU.gif"
0140 BORDER=0 HEIGHT=28 WIDTH=378></A></P>
0150
0160 <H2>This is a test </H2>
0170 </BODY>
0180 </HTML>
```

Note that the actual value of #TITLE was substituted on line 0030. When the output is in the editor source area, HNANCGIP reads all lines and puts them to the browser.

The output written to the browser looks like the following:

```
<HTML>
<HEAD>
<TITLE> This is a Natural Script test </TITLE>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=iso-8859-1">
<META NAME="AUTHOR" CONTENT="A.N. Other">
<META NAME="FORMATTER" CONTENT="Microsoft FrontPage 2.0">
<META NAME="GENERATOR" CONTENT="Mozilla/3.01Gold (WinNT; I) [Netscape]">
</HEAD>
<BODY TEXT="#000000" BGCOLOR="#FFFFFF" LINK="#0000EE" VLIINK="#551A8B"
  ALINK="#FF0000">

<P><IMG SRC="../images/abc.gif" HEIGHT=28 WIDTH=378><BR>
<A HREF="/imagemap/barmenu.map"><IMG ISMAP SRC="../images/BARMENU.gif"
BORDER=0 HEIGHT=28 WIDTH=378></A></P>

<H2>This is a test </H2>
</BODY>
</HTML>
```

## Additional Notes on Natural

The following additional hints and tips may prevent problems while using the SMARTS CGI extensions with Natural:

1. Do not present the first element of a Natural data area structure as a parameter to these routines. Natural will present each element of the structure as a single parameter rather than simply passing a pointer to the structure. Allocate the entire area, redefine the field, and name the main variable as the parameter to the function.

2. Certain functions can result in system ABENDs that terminate the Natural session. Normally, this only happens when certain functions are used in an invalid fashion but are not trapped by Natural abnormal termination routines.

## C Considerations

The SMARTS Software Developer Kit (SDK) is required to support CGI programs based on C.

### Compiling and Linking C Applications

C application programs must be linked using the standard process that would be used to compile a batch program for the operating system where the program will run. The one difference is that the APSvrs.SRCE dataset provided with the SMARTS SDK must be specified instead of the C header library provided with the compiler or operating system.

As SMARTS SDK does not provide or support all of the 800 or so UNIX function interfaces available at present, there may be occasions when it is possible to use the header files and implementation provided with the operating system. Whether this works depends on the function and how closely it interacts with the operating system. Software AG recommends that you first write and run a small program under SMARTS SDK to determine if the functionality will cause a problem.

It is intended that SMARTS SDK will support most if not all of these interfaces in the future. If you need support for a function interface that is not currently supported, contact your Software AG technical support representative for information about when and how the function will be supported. Refer to the SMARTS SDK Programmer's Reference Manual for information about the functions that are currently supported.

### Supplied C Sample Programs and Jobs

The C samples for HTTP processing that are provided on the HTPvrs.SRCE dataset are described in the following table.

Member	This is a sample ...
HCANSAMP	C program to accept a simple CGI request and return some data to the user. When compiled and linked, it may be used in conjunction with the source members HHANCGET or HHANCPUT, which contain HTML and are provided on the HTPvrs.SRCE dataset. Note: PJBSCC and associated link jobs may be used to compile and link this sample if required.
HHANCGET	HTML page that drives the HCANSAMP C program using a form and the HTTP GET method. It may be referenced using the following URL. Refer to the installation verification procedure for information about interpreting this URL. <a href="http://ip-addr:port/htpvrs/srce/hhancget.htm">http://ip-addr:port/htpvrs/srce/hhancget.htm</a>
HHANCPUT	HTML page that drives the HCANSAMP C program using a form and the HTTP POST method. It may be referenced using the following URL. Refer to the installation verification procedure for information about interpreting this URL. <a href="http://ip-addr:port/htpvrs/srce/hhancput.htm">http://ip-addr:port/htpvrs/srce/hhancput.htm</a>

## SMARTS and stdin, stdout, and stderr

The standard I/O files are all supported by SMARTS as follows:

File	Standard Processing	CGI Processing
stdin	stdin is empty and any attempt to access it sets the end-of-file condition.	When the request was generated using the POST method, stdin contains the content data for the CGI request. When the GET method is used, stdin is empty and returns end-of-file, if accessed.
stdout	Output to stdout is written to the 'stdout' DD/DLBL which is allocated either explicitly in the SMARTS procedure or by default as a spool file.	Output to stdout is taken as response data to the CGI request and passed on to the requesting client.
stderr	Output to stderr is written to the 'stderr' DD/DLBL which is allocated either explicitly in the SMARTS procedure or by default as a spool file.	Same as for standard processing

## C and the SMARTS CGI Extensions

The SMARTS CGI extensions were designed with non-C applications in mind. It is not envisaged that C application programs will use these extensions.

## COBOL Considerations

COBOL programs must be compiled and linked as if for the batch environment on the operating system where they will run. Once compiled and available to the SMARTS address space or partition, COBOL programs may simply be run as documented earlier in this manual.

## Sample Programs and Jobs

The following table describes the various HTTP server COBOL samples that are provided on the HTPvrs.SRCE dataset.

Member	This is a sample ...
HHANCOBT	HTML page that drives the HOANSAMP COBOL program using a form and the HTTP GET method. It may be referenced using the following URL. Refer to the installation verification procedure for more information about interpreting this URL. <a href="http://your.ip.address:port/htpvrs/srcce/hhancobt.htm">http://your.ip.address:port/htpvrs/srcce/hhancobt.htm</a>
HJBNCOBC	JCL member to compile and link the sample COBOL CGI program HOANSAMP.
HOANCONV	COBOL program that uses the conversational features of the HTTP server to enable it to converse with a WWW browser over a series of HTML pages. It may be started using the following URL. Refer to the installation verification procedure for more information about interpreting this URL. <a href="http://ip-addr:port/cgi/hoanconv">http://ip-addr:port/cgi/hoanconv</a>
HOANSAMP	COBOL CGI program that is driven by the HHANCOBT HTML page.

## COBOL and the SMARTS CGI Extensions

COBOL programs may use the SMARTS CGI extensions by issuing a call to the appropriate extension module as follows:

```
call 'HAANUPR' using HTPCGI-STATUS, function, parm1, parm2, <..>, parmn.
```

The following calls may also be issued but are only included for compatibility with previous releases. The HAANUPR interface module implements the recommended interface; the following interface requests will not be enhanced:

```
call 'HAANCGIG' using HTPCGI-STATUS, field, length, value.
call 'HAANCGIL' using HTPCGI-status, token entries, name-length,
    name1, value-length1, value1, type1,
    name2, value-length2, value2, type2,
    <..>
    namen, value-lengthn, valuen, type2.
call 'HAANCGIP' using HTPCGI-STATUS, data, length.
call 'HAANCGIT' using HTPCGI-STATUS, data, length.
```

The parameters required by these interfaces are described in SMARTS CGI Extensions .

The HTPCGI-status must be defined as follows in COBOL:

```
01 HTPCGI-STATUS.
   03 HTPCGI-RETURN-CODE pic 9(4)COMP.
   03 HTPCGI-REASON-CODE pic 9(4)COMP.
```

The following describes the COBOL format required for the other parameters:

Name	COBOL Format	Description
field	pic x(<length>)	Must be an alpha field ending in a blank character
variable	pic x(<length>)	Must be an alpha field ending in a blank character
value	pic x(<length>)	Must be an alpha field of length 'length'
length	pic 9(4) COMP	Must be a two-byte binary field containing a length
data	no defined format	The start of an area (normally alpha) of length 'length'

## PL/1 Considerations

PL/1 programs must be compiled and linked as if for the batch environment on the operating system where they will be running. Once compiled and available to the SMARTS address space or partition, PL/1 programs may be run as documented earlier in this manual.

### Sample Programs and Jobs

The following table describes the HTTP server PL/1 samples that are provided on the HTPvrs.SRCE dataset.

Member	This is a sample ...
HHANTPL1T	HTML page that drives the HPANSAMP PL/1 program using a form and the HTTP GET method. It may be referenced using the following URL. Refer to the installation verification procedure for more information about interpreting this URL. http://ip-addr:port/htpvrs/srce/hhancget.htm
HJBNPLIC	JCL to compile and link the sample PL/1 CGI program HPANSAMP.
HPANSAMP	PL/1 CGI program that is driven by the HHANPL1T HTML page.

## PL/1 and External Module Names

Because PL/1 cannot support external names longer than seven (7) characters, aliases are required for the SMARTS extension interfaces. All future interface names will be at most seven (7) characters long.

Aliases are created by changing the first four (4) characters of any extension module name to PL1. Users must create these aliases themselves if they wish to use the functionality.

The following table provides a cross reference between the extension program name as documented in the SMARTS SDK Programmer's Guide and what must be used by PL/1:

PAANATOE	PL1ATOE
PAANETOA	PL1ETOA

These external modules must also be declared to the PL/1 program as follows:

```
DCL PAANHLL EXTERNAL ENTRY OPTIONS(ASM INTER);
DCL PL1ATOE EXTERNAL ENTRY OPTIONS(ASM INTER);
DCL PL1ETOA EXTERNAL ENTRY OPTIONS(ASM INTER);
DCL HAANUPR EXTERNAL ENTRY OPTIONS(ASM INTER);
```

Finally, for PL/1 to load these program dynamically, the following statements must be included:

```
FETCH PAANHLL;
FETCH PL1ATOE;
FETCH PL1ETOA;
FETCH HAANUPR;
```

If these statements are not included, the PL/1 compiler expects them to be linked with the application program, which is not recommended.

## PL/1 and the SMARTS CGI Extensions

PL/1 programs may use the SMARTS CGI extensions by issuing a call to the appropriate extension module as follows:

```
CALL 'HAANUPR' using HTPCGI-status, function, parm1, parm2, <..>, parmN.
```

The calls documented in the other sections have not be included here as no PL/1 applications have been created with older versions of SMARTS.

The parameters required by this interface is described in the section SMARTS CGI Extensions .

The HTPCGI-status must be defined as follows in PL/1:

```
DCL 1 HTPCGI-STATUS,
    2 HTPCGI-RETURN-CODE FIXED BINARY(15),
    2 HTPCGI-REASON-CODE FIXED BINARY(15);
```

The following describes the PL/1 format required for the other parameters:

<b>Name</b>	<b>PL/1 Format</b>	<b>Description</b>
field	CHAR(<length>)	Must be an alpha field ending in a blank character
variable	CHAR(<length>)	Must be an alpha field ending in a blank character
value	CHAR(<length>)	Must be an alpha field of length 'length'
length	FIXED BINARY(15)	Must be a two-byte binary field containing a length
data	no defined format	The start of an area (normally alpha) of length 'length'

## S/390 Assembler Considerations

### Assembler and the SMARTS CGI Extensions

Software AG recommends that Assembler programs use the standard SMARTS API to handle CGI requests.