FIDDLE – Automated creation of gSOAP-ACMS interfaces

User Guide

Author: Brett Cameron
Hewlett Packard

Date: 8 May 2010

Version History

Version	**Date**	**Revised by**	**Description**
1.0	12-Jun-2009	Brett Cameron	Initial draft
1.1	20-Jun-2009	John Apps	Assorted refinements
1.2	22-Jul-2009	Brett Cameron	Added notes on `/names` qualifier
1.3	10-Aug-2009	Brett Cameron	Added notes on tuning
1.4	16-Sep-2009	Brett Cameron	Adding details of new features
1.5	20-Dec-2009	Brett Cameron	Added `/tasks` qualifier
1.5	08-Jan-2010	Brett Cameron	Factored in changes suggested by John Apps
1.6	08-May-2010	Brett Cameron	Added notes for `/records` qualifier
1.7	01-Nov-2010	Brett Cameron	Added brief comment about exit handler

Table of Contents

1. Introduction

`Fiddle` is a simple tool designed to parse STDL files created from ACMS applications and generate code that can be used to streamline the development of interfaces to the ACMS application environment. The tool was initially developed to generate code that could be used with TPware, but it has now evolved to provide a powerful facility to generate a gSOAP-based Web Services wrapper for an ACMS application.

The `fiddle` tool is designed in a modular fashion which makes it reasonably straightforward to incorporate new code generation options or to enhance existing code generation capabilities to include additional functionality to address specific customer requirements.

It is likely that technology options other than gSOAP will be provided in future releases of the software; however, at this time only the gSOAP option is supported.

The following text briefly describes the design of the tool and the code generation options that are currently available.

2. Use

The `fiddle` tool provides a simple OpenVMS CLI-based user interface. To use the tool, it is simply a matter of defining a foreign command for the `fiddle` program as shown below, after which the command qualifiers described in the following section may be used to generate output in the desired format:

```
$ fiddle :== $gsoap$root:[bin]fiddle.exe
```

Note that the logical name `gsoap$root` must be defined as described in the gSOAP for OpenVMS installation instructions.

3. Options

Assuming that the foreign command "`fiddle`" has been defined as shown above, the following sections describe the commands that are currently available.

3.1. *fiddle/version*

This command will simply display the version details of the program. The output includes the version number and the date and time of the build. The `fiddle/version` command takes no arguments and does not accept any command qualifiers.

3.2. *fiddle/gsoap [qualifiers] <stdl-file>*

This command will generate code that can be used to construct a gSOAP-based interface to an existing ACMS application. It is assumed that all tasks have the no terminal I/O option specified. The supplied STDL file is parsed by `fiddle` and a gSOAP-compliant interface definition is produced. The gSOAP interface definition may then be used to create either a stand-alone gSOAP server, or a shareable image that can be used with Apache and `mod_gsoap`. The command provides a number of qualifiers that can be used to control various aspects of the generated gSOAP interface definition; these qualifiers are described below.

3.2.1. Qualifiers

- `/lowercase`

  All identifiers will be converted to lowercase where it is permissible to do so. Identifier names in the STDL file are all uppercase, which may not be appropriate or desirable for the resultant Web service. The `/lowercase` qualifier can be used to instruct `fiddle` to convert identifier names to lowercase, which when used in conjunction with the `/mangle` qualifier (see below) typically produces a more desirable result. HP recommends specifying both `/lowercase` and `/mangle`.

- `/mangle`

  When used in conjunction with `/lowercase`, the `/mangle` qualifier will cause identifiers in the input STDL file to be converted to "camel-case", where it is permissible to do so. All underscore characters are removed from identifier names. The use of the `/mangle` qualifier in conjunction with `/lowercase` typically results in a Web service and service definition (WSDL) that it more amenable to use with other Web services technologies such as .NET and Apache AXIS. While gSOAP does not object to underscores in identifier names, it can cause some issues from a development perspective, and the presence of underscores can lead to unpredictable results with some other Web service technologies. Accordingly, HP recommends the use of the `/mangle` qualifier (in conjunction with the `/lowercase` qualifier) to eliminate underscore characters from identifier names.

  Note: task names and ACMS application names are left unchanged where they are used as input to ACMS Systems Interface routines.

- `/hooks=<language>`

  Instructs `fiddle` to include calls to hook functions in the generated code. If the `/hook` qualifier is specified, `fiddle` will include in the generated gSOAP/ACMS interface code calls to the API function `GSOAP$ACMS_HOOK()`. These calls are included immediately before and after the ACMS task call for each Web service method. The calls are bracketed by pre-processor directives (`#define` directives) so that they can optionally be included or excluded when compiling the generated code. The inclusion of these calls provides a flexible means for developers to incorporate task-specific custom processing requirements with the generated Web service code. For example, the generation of response time statistics for each ACMS task call.

  In addition to including calls to hook functions in the generated code as described above, the `/hooks` qualifier will also cause `fiddle` to generate a template hooks library and a template link options file. The template library code may be generated in either C (`/hooks=c`) or COBOL (`/hooks=cobol`), and includes a template hook function for each ACMS task. The template link options file includes the symbol vector and other such details required to link the hooks library into a shareable image that can be used by the associated gSOAP/ACMS interface environment.

  A detailed description of the hook mechanism and the generated template files is provided elsewhere in this document.

- `/import=<file>`

  The `/import` qualifier can be used to specify the name of a file to be included in the generated interface definition via the gSOAP interface definition `#import` statement. This facility may be used for example to include details of a user-defined SOAP header data structure, which may then be manipulated using the hook function facility.

- `/configuration=<configuration-file-name>`

  This qualifier may be used to specify the name of the configuration file to be used by the interface application.

  As described in section 5, Logical names, it is possible to use a logical name definition to specify a configuration file for the application. However this mechanism is not appropriate in situations where there are multiple interface applications managed by the same instance of Apache as it is not possible to use the logical name definition to specify a different configuration file for each such application. The `/configuration` qualifier provides a means of overcoming this difficulty by allowing developers to explicitly specify a configuration file for each application. The specified configuration file name may be a file name or a logical name that translates to the appropriate configuration file name. The use of a logical name affords greater flexibility.

When the `/configuration` qualifier is used, several additional lines of code are generated to coordinate loading of the specified configuration file. These additional lines of code are bracketed by pre-processor directives, as illustrated below:

```
#ifdef USE_CONFIG
/* Deal with configuration file */
#endif
```

In order to compile in the additional code that manages loading of the configuration file, it is therefore necessary to ensure that the macro `USE_CONFIG` is defined. This would most usually be achieved using the C compiler `/define` qualifier.

- `/identification=<method-name>`

  This qualifier can be used to instruct `fiddle` to include in the generated Web service a method that returns information about the version of the interface, including the version number and UUID taken from the input STDL file, and the date and time of the build. This information is returned by the method in a single string. The method takes no input parameters. If the `/identification` qualifier is not specified, no "identification method" will be generated.

- `/prefix=<namespace-prefix>`

  This qualifier instructs `fiddle` to prepend the specified prefix to all relevant identifiers (such as structure and method names) in the generated gSOAP interface definition file and the gSOAP/ACMS wrapper. If no namespace prefix is specified, the default prefix "`ns`" will be used. In general there is no need to override the default value[1].

- `/service=<service-options>`

  The `/service` qualifier can be used to specify various attributes for the Web service, including the style of the service, and the name and location of the service. Multiple service options can be specified as a comma-separated list, enclosed in parentheses (see examples elsewhere in this document).

  - `style=<service-style>`

    The `style` keyword can be used to specify *style* for the entire Web service. Permitted values are listed below. Note that if no style is specified, "`document`" style will be assumed.

    - `rpc`

      RPC style will be used for the service. The use of RPC style is typically not recommended, and this option should only be specified if RPC style is specifically required by client applications that will be using the service. It should also be noted that the use of RPC style causes `soapcpp2` to generate considerably larger and more complex C/C++ files (`soapC.c`, `soapClient.c`, and `soapServer.c`)

[1] As a general comment, be sure to avoid specifying an empty (zero-length) string for the namespace-prefix or any other option described in this section.

that in some instances (where the STDL includes a large number of tasks) can be too large to compile.[2]

  - `document`

    Document style will used for the service. For reasons outlined above (and in fact for several other reasons), "`document`" style is typically to be preferred over "`rpc`" style; however in most instances it is acceptable to just leave the style unspecified when running `fiddle`, and to manage such matters via the specification of appropriate command line arguments to `soapcpp2`.

- `encoding=<encoding-type>`

  The `encoding` keyword can be used to specify the Web service encoding to be specified for the service in the generated interface definition. Permitted values are listed below, with the default being "`literal`".

  - `encoded`
  - `literal`

  HP recommends using "`literal`" (the default), if possible, as this will typically yield slightly better performance.

  The reader should refer to section 10.1 of the gSOAP User Guide (SOAP RPC Encoding Versus Document/Literal xsi:type Info) for a discussion of the relative merits of SOAP RPC encoding and document/literal styles.

- `name=<service-name>`

  This option can be used to specify the name of the Web service. If this option is not used to specify a name for the service, the name of the service will be derived from the name of the STDL file (possibly converted to lowercase or "camel-case", depending on whether the `/lowercase` and `/mangle` qualifiers have been used).

- `namespace=<namespace-URI>`

  This option can be used to specify the namespace URI for the Web service, where the supplied value (`namespace-URI`) is the URI associated with the namespace prefix. If this option is not specified, a default value will be derived based on the name of the STDL input file. While the value of this URI is not necessarily important for the correct operation of the service, the generated default URI will not be particularly meaningful and should generally be replaced with a more appropriate value.

- `location=<URL>`

  This option can be used to specify the location of the Web service (the URL that would be used by consumers of the service). If no location is specified, then a default location will be derived by `fiddle` using the TCP/IP host name (or address) of the machine on which `fiddle` was run and the username of the user who ran it. Having the correct URL specified in the generated WSDL can simplify the testing, deployment, and use of the resultant Web service.

- `soapaction`

[2] Specifically, for large and complex input STDL files containing a large number of tasks (200+), the size and complexity of the C files generated by `soapcpp2` can be such that the compiler cannot allocate sufficient virtual memory to perform the compilation. Note that this inability to allocate sufficient virtual memory is not necessarily a quota-related problem, but is related to the fact that the C compiler is a 32-bit image, and to how OpenVMS divides up address space.

Specifying this keyword in the list of `/service` options will instruct gSOAP (when generating a SOAP 1.1 interface) to generate a service that uses (or expects to use) the HTTP `SOAPAction` header to resolve service methods.

- `/schema=<schema-options>`

  This qualifier can be used to control the schema-related attributes for the Web service that `fiddle` includes in the generated interface definition. The qualifier is currently of minimal use, supporting only one option (described below); however this facility may be extended in the future to include other options.

  - `namespace=<namespace>`

    This option can be used to specify the schema namespace URI associated with the namespace prefix for the service. The supplied value (`namespace`) should not be prefixed with `urn:` as this prefix is automatically pre-pended by `fiddle` to the supplied value. Note that this option is not particularly relevant to services generated using `fiddle`, and there is typically little point in overriding the default value.

- `/names=<names-option>`

  This qualifier can be used to control how parameter names for task arguments are constructed by `fiddle`. Permitted options are "`types`" and "`derived`", with "`derived`" being the default behaviour. If `/names=types` is specified, the names of task parameters will be based on the names of the associated record types by pre-pending the record name with a "p" and appending a 2-digit integer to ensure uniqueness. If the `/names` qualifier is not specified or if `/names=derived` is used, parameter names will be given unique names of the form p*nnnn*, where *nnnn* is a 4-digit integer. The choice of naming convention is of little relevance with regard to the generated interface code; however using "`types`" will yield names that will be more meaningful to developers implementing client application code.

- `/records=<record-names-option>`

  This qualifier can be used to control how nested record structure types are named by `fiddle`. Permitted options are "`derived`" and "`names`", with "`derived`" being the default behaviour. If `/records=names` is specified, names for nested record structure types are derived by concatenating the names of the top-level parent record structure and the field name of the nested structure. A 2-digit number is appended to the end of the name to ensure uniqueness. If the `/records` qualifier is not specified or if `/records=derived` is used, nested record structure types are named by concatenating the top-level parent record name and a string of the form "`Struct`*nn*", where *nn* is a unique 2-digit integer value (the numbering sequence is local to the record in question). The choice of naming convention is of little relevance with regard to the generated interface code; however using "`names`" will yield record type names that will be more meaningful to developers implementing client application code.

- `/tasks=<task-list>`

  This qualifier can be used to specify a specific list of tasks for which interface code should be generated. In some instances, it may not be desirable or necessary to generate interface code for all tasks defined in a particular STDL file, and in such instances the `/tasks` qualifier may be used to specify only those tasks that are of interest. It should be noted that the generated interface code defines a web service containing only methods and record structures definitions applicable to the specified tasks. The `/tasks` qualifier may also be useful during development, by allowing developers to focus on specific tasks and to reduce build times (which can be substantial for large STDL files).

3.2.2. Parameters

The `fiddle/gsoap` command accepts a single parameter, specifying the name of the STDL file that defines the ACMS interface for which the Web service interface is to be generated. The specified STDL would typically be derived directly from ACMS using the `ACMSADU` utility. However, there is nothing to prevent developers from creating the STDL by other means, or from modifying STDL files derived from ACMS. For example, in some instances it may not make sense to include all of the tasks for a particular ACMS application in the generated Web service. In such cases the STDL obtained from ACMS may be edited as required to include only the relevant tasks and associated record structures. An example of this may be tasks which contain the exchange I/O option and are therefore not supported by `fiddle`.

3.2.3. Example

The following simple example illustrates basic use of the `fiddle` command. The example uses the `/lowercase` and `/mangle` qualifiers to "camel-case" identifiers, and the `/service` qualifier is used to specify the location of the service and the type of encoding to be used for the service. The `/hooks` qualifier instructs `fiddle` to include hook function calls in the generated interface code and to generate template hook library and link options files.

```
$ fiddle/gsoap/lower/mangle/hooks -
/service=(encoding=literal, -
location="http://16.41.224.180/emp") employee_info_appl_wsi.stdl
```

The files that will be created by `fiddle` by running the above command are summarised in the following table.

Filename	Description
`employee_info_appl_wsi.h`	The gSOAP interface definition to be used as input to `soapcpp2`.
`employee_info_appl_wsi.c`	The generated gSOAP-ACMS interface code for the Web service.
`employee_info_appl_wsi.cob_template`	Template hook library code (note that this is COBOL code).
`employee_info_appl_wsi.opt_template`	Template link options to assist with building the hook library.

4. Overview of the development process

To be done!

5. Logical names

This section describes logical name definitions that can be used to specify and control various characteristics of the run-time environment for HP-developed gSOAP components (including the gSOAP-ACMS interface environment used by `fiddle`-generated code). For example, there are logical names that can be used to enable debug logging and to specify the name of the log file to be used by an application, and there are logical names to control aspects of the ACMS environment, such as the user name used to sign in to ACMS, and the default ACMS application name.

5.1. *Generic logical names*

The following table lists generic logical names that are used all HP-developed gSOAP components. As noted previously, these logical names may be defined at the system, group, or process level. If the logical names are concurrently defined at multiple levels, the process-level definition will take precedence over the group-level definition, which in turn will take precedence over the system-level definition.

Logical Name	Description
`GSOAP$DEBUG`	Defining this logical name (to anything) will enable debug logging for HP-developed gSOAP components, including the gSOAP-ACMS interface. Debug output will be written to the application log file if a log file has been specified; otherwise output will be written to `SYS$ERROR`.
`GSOAP$TRACE`	This logical name is not currently used; however it may be used in future releases to enable the tracing of function calls within HP-developed gSOAP components.
`GSOAP$LOG_FILE`	If this logical name is defined and the definition corresponds to a valid OpenVMS file name, then all debug output and operational messages produced by the HP-developed gSOAP components will be written to this file. The logical name definition is ignored (overridden) if a log file is specified in the configuration file used by the application (see elsewhere in this document for a description of the configuration file). If the specified log file name includes a valid integer or hexadecimal C-style format specifier, this will be replaced with the PID of the running process.

5.1.1. Example (log file specification)

As noted in the above table, the logical name `GSOAP$LOG_FILE` can be used to specify the name of the log file to be used by an application for all debug output and operational messages produced by HP-developed gSOAP components, and if the specified file name includes a valid integer or heximdecimal C-style format specifier, then this will be replaced by the PID of the running process.

Consider the following logical name definition:

```
$ DEFINE GSOAP$LOG_FILE SYS$LOGIN:GSOAP_%d.LOG
```

In this particular instance, the file specification includes the `%d` format specifier, which will be replaced by the decimal representation of the PID. For example, if the PID is 000058DB, then the resultant file name would be `SYS$LOGIN:GSOAP_22747.LOG`.

Arguably it is more useful to have the hexadecimal representation of the PID included in the file name, and this may be achieved using the `%x` format specifier (or some variation thereof) as follows:

```
$ DEFINE GSOAP$LOG_FILE SYS$LOGIN:GSOAP_%08X.LOG
```

5.2. *ACMS/gSOAP logical names*

The following table lists logical names that are specifically used by the HP-developed ACMS/gSOAP components. As with the generic logical names described in the previous section, the logical names tabulated here may be defined at any level, and the same precedence considerations apply when logical names are concurrently defined at multiple levels.

Logical Name	**Description**
`GSOAP$ACMS_USER`	This logical name can be used to define the ACMS username that is to be used by the gSOAP-ACMS interface when signing in to ACMS. This logical name definition is ignored if an ACMS username is specified in the application configuration file. If a username is explicitly specified for a standalone gSOAP server (typically via a command line option), this will override both the logical name definition and the configuration file setting, and if no username is specified via any of these three mechanisms, the username associated with the account under which the process is running will be used.
`GSOAP$ACMS_APPLICATION`	Code generated by `fiddle` uses the ACMS application name (package) specified in the input STDL file, and by default the gSOAP-ACMS interface uses the application name from the STDL file for all task calls. The interface can be instructed to use an alternative ACMS application by defining the logical name `GSOAP$ACMS_APPLICATION` with the name of the application in question. Note that this is a global setting, with all task calls going to the same ACMS application. If a configuration file is used, it is possible to associate tasks with specific applications on a per-task basis (see elsewhere in this document for a description of this facility). Any such options specified in the configuration file take precedence over the logical name definition, which in turn takes precedence over the application name taken from the STDL file.
`GSOAP$ACMS_HOOKS`	This logical name can be used to specify the name of an OpenVMS shareable image containing “hook” functions for a gSOAP/ACMS application. If the `/hook` qualifier was used when generating the gSOAP-AMS interface code and the generated code was compiled with the `HOOKS` macro defined, then the function `GSOAP$ACMS_HOOK()` will be called immediately before and immediately after the ACMS task call. If a shareable image has been specified using the `GSOAP$ACMS_HOOKS` logical name, `GSOAP$ACMS_HOOK()` will attempt to resolve and invoke developer-written functions in the shareable image, based on the ACMS task name. Note that the logical name definition `GSOAP$ACMS_HOOKS` is ignored (overridden) if a hook functions shareable image is specified in the configuration file used by the application

	(see elsewhere in this document for a description of the configuration file). A detailed description of the hook functions mechanism is provided elsewhere in this document; however in general terms, this facility provides a means for developers to customise the operation of the generated gSOAP-ACMS interface.
`GSOAP$ACMS_CONFIG`	The `GSOAP$ACMS_CONFIG` logical name can be used to specify the name of the configuration file to be loaded by the application.
`GSOAP$ACMS_TIMEOUT`	This logical name can be used to define a global timeout value for ACMS task calls. The timeout value is specified in seconds and the timeout value must be between 10 seconds and 300 seconds (inclusive). This logical name definition is ignored if a timeout value is specified in the application configuration file using the `set timeout` command (see Section 7), or if a specific timeout value has been specified for a task using the `set task/timeout` command. If no timeout value is specified for the application, a default timeout value of 60 seconds will be used.
`GSOAP$ACMS_DATE_FORMAT`	This logical name can be used to specify the textual date format to be assumed by the gSOAP-ACMS interface code when converting dates to and from OpenVMS binary date format. The value specified for the logical name should be a quoted format string containing zero or more conversion specifications and ordinary characters, as supported by the C run-time library functions `strftime()` and `strptime()`. The definition of this logical name is ignored if a date format is specified in the application configuration file using the `set format/date` command (see Section 7). If no date format is specified for the application, the standard OpenVMS date format (`DD-MMM-YYYY hh:mm:ss`) will be used, which corresponds to a format specification value of `"%d-%b-%Y %H:%M:%S"`. If an invalid format is specified, a warning will be logged, and the default format will be assumed.
`GSOAP$ACMS_SNDOPR`	Defining this logical name (with any value) will cause the gSOAP-ACMS interface to send a message to operator terminals (`OPCOM CENTRAL`) when the interface process exits. It is recommended that this logical name should only be defined in production and test environments. Note that there is no equivalent configuration file command for this logical name.

5.3. *Important note*

As a general comment, it should be noted that the use of many of the logical names described above may not be appropriate when running gSOAP-ACMS applications under Apache (using `mod_gsoap`), as the logical names are generally quite application-specific. For example, it will typically be necessary for each gSOAP-ACMS application to have its own configuration file, log file, and so on. In order to facilitate this when running multiple applications under the same Apache instance, it is necessary to use the `fiddle /configuration` qualifier (see section 3.2.1) to specify a unique configuration file for each application.

6. Hook functions

Hook functions provide developers with the ability incorporate additional functionality into gSOAP-ACMS interface applications on a task by task basis without having to modify the generated interface code. Calls to hook functions are included in the `fiddle`-generated interface code when the `/hooks` qualifier is used for the generation of the interface. Calls to hook functions are included in the generated code immediately before and immediately after calls to ACMS tasks. Typical uses of hook functions would include pre and post-processing of ACMS task parameters, customised logging, and the incorporation of caching facilities.

Hook functions must have the same name as the ACMS task they are associated with. The functions are linked together into a shareable image with the hook functions specified in the symbol vector. This shareable image is referred to as a "*hook library*". At runtime, if a hook library has been specified (via the logical name `GSOAP$ACMS_HOOKS` or using the "`set hooks`" configuration file command), the gSOAP-ACMS interface code will load the library and the generated Web service method code will attempt to resolve the address of the associated hook function. If the address of a particular hook function is resolved then the address is saved for subsequent re-use and the hook function is called by the interface code before and after the ACMS task call. If no hook function is available for a particular task, this information is similarly saved for subsequent re-use so that the interface code does not repeatedly try to resolve the function address every time the associated Web service method is called. If an error was encountered trying to resolve a particular hook function, the error will be reported and no further attempts will be made to resolve the function.

As commented previously, hook functions are only resolved when they are first needed, and the location (address) of the functions are saved for subsequent re-use, thereby avoiding the computational overhead of repeatedly resolving the addresses of hook functions each time they are required.

The function prototype of a hook function is illustrated below (using C language syntax), and the function parameters are described in the subsequent test.

```
unsigned long MY_HOOK(
        unsigned long *mode, struct soap *soap, unsigned long *ud, ...)
```

- `mode`

  The observant reader will have noticed that there is only one hook function for each ACMS task (and the associated Web service method), as opposed to there being a separate hook function for pre-task and post-past processing. The first parameter passed to a hook function is an unsigned longword "`mode`" parameter, which can be used by the hook function to determine whether the function has been called before or after the ACMS task, whether errors have been encountered, and so on. The following table summarises the various `mode` values and their meaning:

Mode	Description
0	Pre-task call.
1	Post-task call. The implication is that the scenarios associated with modes 2 and 3 below have not occurred.
2	Post-task call indicating the data was found in cache by the pre-task hook

	call and the ACMS task call has been bypassed.
3	Post-task call indicating that either the pre-task hook function call or the ACMS task call retuned an error.
Other	Any other `mode` values should be treated as errors; however in theory such situations cannot occur.

As noted previously, the mode parameter is an unsigned longword value, and it is passed by reference to the hook function. The hook function should not modify this value; however doing so will have no effect on subsequent processing performed by the associated gSOAP-ACMS interface code.

- `soap`

  The second hook function parameter is the address of the gSOAP SOAP structure for the interface application. This structure stores the gSOAP runtime environment, and it can be used (either directly or via various gSOAP API functions) to access and modify numerous characteristics of the gSOAP runtime environment[3]. For most applications, it will not be necessary for hook functions to do anything with this structure; however there are some situations where hook functions having access to gSOAP runtime can be useful. For example, a user-defined SOAP header structure may be included in the fiddle-generated gSOAP interface definition using the `/import` qualifier, and the SOAP header data may then be accessed and manipulated in hook functions via the gSOAP runtime environment structure.

  Note that when working with COBOL hook functions, the address of the gSOAP runtime environment structure is supplied in an unsigned longword, which must be passed by value to other functions.

- `ud`

  The third parameter passed to the hook function is the address of an unsigned longword variable that may be used by the hook function to store information that may need to be passed between pre and post hook calls. In most cases where such functionality is required, the parameter will be used to store the address of a piece of data.

- `...`

  The third and subsequent parameters passed to the hook function are the addresses of any ACMS task parameters.

Hook functions must return a valid OpenVMS unsigned longword status code value. Under normal circumstances, the status `SS$_NORMAL` should be returned. If an error occurs, a relevant fatal or severe error status should be returned by the hook function, and the message text associated with the error status will be logged by the interface code and returned by the Web service to the caller as a SOAP error. In addition, pre-task hook calls may also return a special informational status code (`GSOAP$_INCACHE`) to indicate to the interface code that the ACMS task call should be bypassed. This status code would typically be returned by the pre-task hook call in situations where some form of caching is being used and the requested data has been found in cache by the pre-task hook call, thereby negating the need for the interface code to call the ACMS task to retrieve the data in question.

The global constant `GSOAP$_INCACHE` may be resolved by linking hook function code with the gSOAP object library (`GSOAP.OLB`, or one of its variants). If for any reason it is not feasible to

[3] The reader should refer to the gSOAP User Guide for detailed information regarding the gSOAP runtime environment and how it may be manipulated.

link hook function code with the gSOAP object library, hook function code should simply define a local variable to the value of `GSOAP$_INCACHE`, and use this instead. The value of `GSOAP$_INCACHE` is `139427851` (`084F800B`).

6.1. Template hook functions

As noted above, calls to hook functions are included in the `fiddle`-generated interface code when the `/hooks` qualifier is used for the generation of the interface. In addition to including hook function calls in the generated interface code, the `/hooks` qualifier also causes `fiddle` to generate a set of template hook functions in either C or COBOL, with the resultant files being *`appl`*`_hooks.c_template` and *`appl`*`_hooks.cob_template` respectively, where *`appl`* is the name of the ACMS application. As well as generating these template hook function files, a link options file (*`appl`*`_hooks.opt`) is also created, which includes a symbol vector containing all of the hook functions.

The generated hook function template code includes a basic hook function for each task, and can be used as the starting point for developing more useful hook functions. The link options file may be used to help with linking your hook functions into a shareable image that can be loaded and called by the `fiddle`-generated gSOAP/ACMS interface.

Example to be provided!

7. Configuration file format

As discussed previously, various characteristics of the gSOAP-ACMS interface environment such as debug logging, the use of hook functions, and so on, can be controlled via logical name definitions or through the use of a simple configuration file. The configuration file mechanism affords much greater flexibility, and is generally to be preferred over the use of logical name definitions. In addition, the configuration file facility supports configuration options that cannot be facilitated through the use of logical names. Specifically, the configuration file provides a powerful facility to associate specific ACMS tasks with a specific ACMS application, on a per-task basis.

The configuration file facility uses OpenVMS DCL-like command syntax to specify various characteristics of the interface environment. This functionality is provided through the use of the OpenVMS CLD utility and the associated CLI utility routines, and affords considerable flexibility in terms of adding new configuration commands or adding new options to existing commands. The following text summarises the various configuration commands that are currently supported by the configuration file facility. It should be noted that without exception, configuration options specified in a configuration file override the related logical name definitions.

- `set default <acms-application-name>`

  This command can be used to specify the default ACMS application name that is to be used by the interface application when calling tasks for which no specific application name has been specified using the "`set task`" command (described below). This will override the ACMS application name taken from the STDL file used to generate the interface and is comparable to defining the logical name `GSOAP$ACMS_APPLICATION`.

  Note that no checking is performed when the configuration file is loaded as to whether the specified default ACMS application exists and is started. The gSOAP-ACMS interface software is designed so that it can be started before any ACMS applications are started, and in addition it is generally possible to restart ACMS applications without having to restart the interface environment.[4]

- `set debug`

  This command can be used instead of defining the logical name `GSOAP$DEBUG` to enable debug logging for HP-developed gSOAP components, including the gSOAP-ACMS interface environment.

- `set user <user-name>`

  This command can be used instead of defining the logical name `GSOAP$ACMS_USER` to specify the username that is to be used by the gSOAP-ACMS interface environment when signing in to ACMS. As noted previously, a username specified in the configuration file takes precedence over a username specified using the `GSOAP$ACMS_USER` logical name; however if a username is explicitly specified for a standalone gSOAP server (typically via a command line option), this will take precedence over both the logical definition and the configuration file username specification. If no username is specified via any of the mechanisms described, the username associated with the current process will be used by the interface environment when signing in to ACMS.

- `set task <task-name>`
  `/application=<acms-application-name>/timeout=<seconds>`

  The "`set task`" command can be used to associate specific ACMS tasks with a specific ACMS application and to specify specific timeout values for particular tasks. Any number of "`set task`" commands may be specified in the configuration file, and any duplicate entries will be ignored[5]. Any calls by the interface environment to tasks for which no ACMS application has been explicitly specified will simply use the default application name and timeout value.

  Task timeout values should be specified in seconds, and should be between 10 seconds and 300 seconds, inclusive. If a value outside of this range is specified, the timeout value will be adjusted to the minimum or maximum timeout value, as appropriate. If no specific timeout is specified using the `/timeout` qualifier, the default timeout for the application will be used (if a timeout value has been specified via the "`set timeout`" command or using the logical name `GSOAP$ACMS_TIMEOUT` then this value will be used, otherwise the default value of 60 seconds will be used).

  As commented above, any number of "`set task`" commands may be specified. Task and application details are stored in a hash table (keyed on task name) that uses a hashing algorithm optimised for short strings (task names) and sized to comfortably accommodate at least 130 entries with minimal chance of collisions (two or more task names having the same hash value). However, in order to ensure optimal performance of the hash table for task lookups, some care should be taken when using the "`set task`" command to ensure that entries are specified in the configuration file only for those tasks that do not use the default application or timeout (although as a hash table is an O(1) algorithm, this is not a particularly major concern).

- `set log <filename>`

[4] Note that while it is generally possible to restart ACMS *applications* used by the gSOAP-ACMS interface environment without restarting the interface environment, if the ACMS *system* is restarted then it will also be necessary to restart the interface environment.

[5] If a duplicate entry is detected, a message will be written to the log file (or to `SYS$ERROR` if no log file has been specified) stating that a duplicate was detected and has been ignored.

This command can be used instead of defining the logical name `GSOAP$LOG_FILE` to specify the name of the log file to be used by the gSOAP-ACMS interface environment for the logging of all debug output and operational messages. A log file specification in the configuration file takes precedence over a log file specified using the `GSOAP$LOG_FILE` logical name. As discussed in section 5.1.1, if the specified log file name includes a valid integer or hexadecimal C-style format specifier, this will be replaced with the PID of the running process. If no log file name is specified for the interface environment, then all debug output (if debug logging is enabled) and informational messages will be written to `SYS$ERROR`.

- `set hooks <filename>`

  This command can be used instead of defining the logical name `GSOAP$ACMS_HOOKS` to specify the name a shareable image containing "hook" functions for a gSOAP/ACMS application (see section 6). A hook file specification in the configuration file takes precedence over a hook file specified using the `GSOAP$ACMS_HOOKS` logical name.

- `set timeout <seconds>`

  The set timeout command can be used to specify a timeout value to be used for ACMS task calls. The timeout value should be specified in seconds, and the specified timeout value should be between 10 seconds and 300 seconds, inclusive. If a value outside of this range is specified, the timeout value will be adjusted to the minimum or maximum value, as appropriate. Specifying a timeout value in the configuration file will override a timeout value specified by using the `GSOAP$ACMS_TIMEOUT` logical name, and if no timeout is specified by any means, a default timeout of 60 seconds will be used for all task calls.

- `set format/date <format>`

  This command can be used to specify the textual date format to be used by the gSOAP-ACMS interface code when converting dates to and from OpenVMS binary date format. The supplied format value must be a quoted format string containing zero or more valid conversion specifications and ordinary characters, as supported by the C run-time library functions `strftime()` and `strptime()`. Specifying a date format in the application configuration file overrides a date format specified using the `GSOAP$ACMS_DATE_FORMAT` logical name, and if no date format is specified for the application, then the standard OpenVMS date format (`DD-MMM-YYYY hh:mm:ss`) will be used, which corresponds to a format specification of `"%d-%b-%Y %H:%M:%S"`. If an invalid format specification is supplied, a warning will be logged, and the default format will be assumed.[6]

It should be noted that commands specified in the configuration file cannot be split across multiple lines. The commands presently supported by the configuration file facility are quite short and there is therefore no need to include functionality to handle commands spanning multiple lines. This behaviour may be revised in future releases of the software.

If any of the commands "`set hooks`", "`set user`", "`set timeout`", "`set format`", and "`set default`" is specified multiple times in a configuration file, no errors or warnings will be reported, and the last specification will be used. This is indicative of the fact that processing of the configuration file is completed before any ACMS-related operations are performed.

[6] Considerable care should be taken when specifying the date format, as an incorrect format specification may cause applications using the gSOAP/ACMS interface to operate incorrectly.

7.1. Specifying the configuration file

When using the standalone gSOAP Web server facility, it is possible (and arguably most convenient) to specify the name of the configuration file via the `GSOAP$ACMS_CONFIG` logical name; however when running multiple gSOAP-ACMS interface applications under the same Apache instance, it will invariably be necessary to have a separate configuration file for each interface application. In this scenario, it is not feasible to use the `GSOAP$ACMS_CONFIG` logical name, and instead the `/configuration` option (see section 3.2.1) should be used with `fiddle` to explicitly specify a configuration file for each application. The name of the configuration file specified using the `/configuration` qualifier may be either a filename or a logical name that translates to the desired configuration file name. Specifying a logical name for the configuration file when using the `/configuration` qualifier provides greater flexibility, and is to be preferred over a fixed filename.

7.2. Sample configuration file

The following listing for a very simple configuration file illustrates the main features of the configuration file facility. Debug logging is enabled, and will be written to the log file specified with the "`set log`" command; the user name to be used when signing in to ACMS and the default application are defined; and the "set task" command has been used to specify that an alternative ACMS application should be used for the `WSI_GET_EMPL_INFO` task. The "`set format/date`" command is used to set the date format to "`dd/mm/yyyy hh:mm:ss`".

Note that lines beginning with an exclamation mark ("`!`") are treated as comments.

```
! Enable debug logging and log output to sys$login:emp.dat
!
set debug
set log sys$login:emp.log
!
! Set the ACMS user name
!
set user smith
!
! Set date format to dd/mm/yyyy hh:mm:ss
!
set format/date "%d/%m/%Y %H:%M:%S"
!
! set the default application
!
set default EMPLOYEE_INFO_APPL_WSI
!
! Set specific task details
!
set task WSI_GET_EMPL_INFO/application=EMPLOYEE_INFO_APPL_WSI02
!

! All done
```

8. Tuning

In order to achieve optimal performance with the gSOAP-ACMS, it is important to have the ACMS and Apache Web server environments appropriately tuned. Specific details of the tuning that may be required will vary from one application environment to another; however there are a number of important factors that will be relevant to the performance of any gSOAP/ACMS application environment:

1. Ensure that the maximum number of worker process that can be started by Apache matches or exceeds the maximum number of server instances that can be started by ACMS for the ACMS application environment in question. Note that this is a very crude approach, and ideally the most appropriate minimum and maximum number of Apache workers should be determined by taking into careful consideration ACMS and Apache response times, network latencies, the number of simultaneous requests, think times, and so on; however there is no point in having the maximum number of Apache workers set to less than the maximum number of requests that can be simultaneously handled by the backend ACMS environment, and keep in mind that ACMS is also able to queue up requests (within given resource constraints).
2. Ensure that the ACMS `MSS_POOLSIZE` system parameter is set to sufficiently high value. If the value of this parameter is too low, the performance of the gSOAP-ACMS interface may be seriously impacted. The default value for `MSS_POOLSIZE` is 512, which is likely to be too small for any application environment that must support a moderate to high transaction rate. This value should be increased to 8192 or higher.
3. Related to the previous point, if ACMS audit logging is enabled, it is also advisable to ensure that the parameters listed in the table below are set higher than their default values and that the logical names `ACMS$ATL_ALQ_BLOCKS` and `ACMS$ATL_DEQ_BLOCKS` both be defined (at system level) as 65535.

Parameter	**Default value**	**Suggested minimum value**
`WS_POOLSIZE`	256	512
`WSC_POOLSIZE`	128	256
`TWS_POOLSIZE`	1600	3200
`TWSC_POOLSIZE`	50	100

   Failure to have these logical names and parameters set correctly can cause ACMS to periodically stall when operating under high transaction loads. Setting the logical names and parameters as described will ensure that the audit logging facility has sufficient resources available and will help to prevent any such stalls. Note that the above table specifies suggested minimum values for the parameters in question; for some installations somewhat higher values may be required.

The reader should refer to the relevant Apache and ACMS product documentation for details of how to implement the recommendations described above.

9. Outstanding issues

It should be emphasised that `fiddle` is beta software, and feedback on how it may be improved would be greatly appreciated. At the time of writing, the following issues are known to exist with the software.

- Some ACMS/CDD/STDL data types are currently supported, and some data types have not been tested thoroughly. Most data types are properly handled; however issues may be encountered with `OCTET` and other such obscure data types.

- The token `DATA` is treated by `fiddle` as a reserved word in all situations. ACMS and CDD allow the word `DATA` to be used as a field name. If an issue related to this matter is encountered, HP would suggest changing the field name in the STDL file, or preferably in the original CDD definition (although it is appreciated that this may necessitate considerable additional effort).
- While tools such as soapUI generally accept WSDL produced by the gSOAP `soapcpp2` utility from a `fiddle`-generated interface definition, problems have occasionally been observed with both the .NET `wsdl.exe` tool and the AXIS2 `wsdl2java` tool. These issues would appear to be due to problems (limitations) with the .NET and AXIS2 tools, and these problems can typically be overcome by trying different command line options with `soapcpp2`.

10. Error and informational messages

The following table lists the error and informational messages that are specific to the gSOAP/ACMS interface software. In addition to these status codes, the various API functions used by the generated interface may return a range of other status codes, including ACMS codes, and status codes retuned by OpenVMS system service and library calls. The `fiddle`-generated interface code implements robust error checking, and will log any errors that are returned by the gSOAP/ACMS API functions.

Status code	Message text	Severity	Comments
`INCACHE`	Requested data retrieved from cache	`I`	Informational message for use by hook functions to indicate that task response data was found in cache and no task call needs to be performed. Refer to Section 6 above for more information regarding this status code.
`INSFMEM`	Insufficient dynamic memory	`F`	This status may be returned by various API functions, and indicates that dynamic memory could not be allocated from the heap. It is unlikely that processing can continue if this error is encountered.
`INVBLEN`	Invalid buffer length	`E`	This error message may be returned by the interface API function that converts null-terminated C strings into a space padded fixed-length strings. The error indicates that an input C string is longer than the expected maximum length (the string is not truncated). If debug logging is enabled (via the `GSOAP$DEBUG` logical name, or otherwise), the offending string will be written to the log file.
`BADDATE`	Invalid date value	`E`	This error message is returned by date conversion routines when a bad date is encountered. The offending date may be a bad OpenVMS binary date/time value or it may be an invalid date/time string.

11. Other changes

1. The maximum length of the fiddle command string has been increased from 256 to 512 characters. With addition of new features and new command qualifiers, a maximum command length of 256 characters was determined to be insufficient.
2. The gSOAP/ACMS runtime environment now implements an exit handler that attempts to perform various clean-up tasks before the program (for example, an Apache worker process) terminates. These clean-up tasks include signing out from ACMS (if connected), and freeing up any resources associated with gSOAP. While it is not strictly necessary to sign out of ACMS (ACMS will automatically clean up any aborted connections), it is good practice to do so, and failure to sign out properly can result in error messages being reported in the ACMS event log on OpenVMS Alpha. If `GSOAP$DEBUG` is defined then a message will be written to the log file stating that the exit handler was invoked.