SOMTEMPL: The SOM Template Application Builder

The **somtempl** program is a programming utility that generates SOM source files and programs from predefined templates. It is a useful tool to simplify repetitive SOM programming tasks. In general, it can generate any sort of text file from a predefined template; however, it's options are primarily focused at SOM programming. It can also be used similarly to C++ templates to define parameterized SOM classes. For beginners, **somtempl** is also useful because it includes a template for a complete SOM/DSOM program – including make files for AIX, OS/2 and Windows.

Running somtempl

To run **somtempl**, simply enter at the command line:

```
somtempl
```

When run without arguments, **somtempl** displays help on its options and also shows a list of the available templates that can be generated. To generate one of the templates, use the –a option. For example, to generate a generic IDL file from a template named gidl, enter:

```
somtempl –a gidl
```

With this command, the file defined in the gidl template will be generated into the current directory. The gidl template generates the following IDL file named xxxx.idl by default:

```
#ifndef xxxx_idl
#define xxxx_idl
#include <somobj.idl>
#include <somcls.idl>
interface defaultClass : SOMObject
{
  // Attributes filled in here:
  // Operations filled in here:
#ifdef __SOMIDL__
  implementation {
    releaseorder:;
    // Class Modifiers:
    align     = 4;
    dllname   = "xxxx.dll";
    metaclass = "SOMClass";
    // Attribute Modifiers:
    // Overrides:
    somDefaultInit: override, init; // Default object initializer
    somDestruct: override;          // Default object uninitializer
  };
#endif /* __SOMIDL__ */
};
#endif  /* xxxx_idl */
```

When a template is generated, the names of files and classes have predefined values by default. These values can, however, be changed with options given on the **somtempl** command line. These options are described below.

Option

–n *classname*

The template will be generated with the given class name. If no class name is given with this option, the class name defaults to **defaultClass**.

–s *filestem*

This option allows you to uniquely name the files that are generated by **somtempl**. The default file stem is **xxxx**. On systems that limit file names to 8 characters, the filestem given with this option should be limited to 4 characters.

–p *class=filestem*

This option allows you to select a parent class for the main class generated in the template. If no parent is specified, the default parent is **SOMObject**. This is equivalent to –p **SOMObject=somobj**. More than one parent class can be specified by including more than one –p option on the command line. The directories specified in the SMINCLUDE environment variable are searched to locate the specified file named *filestem.idl*. If the file is not found, a new parent class file named *filestem.idl* is generated from the included generic IDL file template.

–m *class=filestem*

This option allows you to select a metaclass for the main class generated in the template. If no metaclass is specified, the default metaclass is **SOMClass**. This is equivalent to –m **SOMClass=somcls**. Only one metaclass option can be specified. The directories specified in the SMINCLUDE environment variable are searched to locate the specified file named *filestem.idl*. If the file is not found, a new metaclass file named *filestem.idl* is generated from the included generic IDL file template.

–e

This option inhibits the searching of the directories named in the SMINCLUDE environment variable. This affects the operation of the –m and –p options.

The following example uses the above options to generate the generic IDL template with a class named “Animal”, a parent class named “SOMPPersistentObject”, a metaclass named “AnimalFactory” and file names beginning with “ani”.

```
somtempl –a gidl –n Animal –s ani –p SOMPPersistentObject=po
–m AnimalFactory=animeta
```

The main IDL file produced by this command, named “ani.idl”, is shown below. Note the differences between this and the earlier generic IDL file. The parent class file “po.idl” already exists in the SOM include directory; therefore, it is not created in the current directory. The “AnimalFactory” metaclass in “animeta.idl” did not previously exist; thus the “animeta.idl” file is generated into the current directory.

```
#ifndef ani_idl
#define ani_idl
#include <po.idl>
#include <animeta.idl>
interface Animal : SOMPPersistentObject
{
  // Attributes filled in here:
  // Operations filled in here:
#ifdef __SOMIDL__
  implementation {
    releaseorder:;
    // Class Modifiers:
    align     = 4;
    dllname   = ”ani.dll”;
    metaclass = ”AnimalFactory”;
```

```
    // Attribute Modifiers:
    // Overrides:
    somDefaultInit: override, init; // Default object initializer
    somDestruct: override;          // Default object uninitializer
  };
#endif /* __SOMIDL__ */
};
#endif  /* ani_idl */
```

Templates may include more than just an IDL file. Supplied with **somtempl** is a DSOM template (option –a dsom). This template includes the IDL files, as well as the C source files and make files to create a complete DSOM application.

Template files

The **somtempl** program uses two files (located via the SMINCLUDE environment variable) to determine the files to be generated and the available templates:

- The first file is called the configuration file. The configuration file contains the names of the templates that can be generated and the names of the files that make up the template. The configuration file also contains user–defined symbols for substitution when files are generated.
- The contents of the files defined by the configuration file are contained in the second file, the template file.

The supplied templates

The default configuration file used by **somtempl** is named **somtempl.cfg**. The default template file is named **somtempl.app**. These files are supplied in SOM's include directory. To change the names of the configuration and template files used by **somtempl**, you can use the following options when you run the program:

Option

–f *configfile*

configfile is the name of the **somtempl** configuration file. Default is somtempl.cfg.

–t *templatefile*

templatefile is the name of the **somtempl** template file. Default is somtempl.app.

Contained in the default configuration and template files, you will find the following templates:

Template

gidl

A template of a generic IDL file. Generate this template and then add your own methods, attributes and data.

idl_dll

A template of the same generic IDL file as in the gidl template plus make files, module definition files and class library initialization source for AIX, OS/2 and Windows. The generated template can be built immediately with the make utility.

dsom

This template generates a complete Distributed SOM program for a basic SOM class. Also generated are the make files, module definition files, class library initialization source and source to simplify local/remote object transparency. The generated application creates either local or remote objects and includes command scripts to set up the server location.

dsomsvr

The template generates a DSOM server program. The program generated can be used in place of the somdsvr program supplied with the SOMobjects Toolkit. Only the –s option is useful when generating this template. The –s <stem> option allows you to change the server program name from xxxxsvr to <stem>svr.

Other somtempl options

–o

This option indicates you will allow **somtempl** to overwrite existing files. By default, a file in the current directory with the same name as a generated file will not be overwritten.

–i *file*

When this option is used, **somtempl** will read options from the named *file*, rather than from the command line.

–u *name=value*

This option specifies that a user–defined symbol *name* should be replaced by the given *value* when files are generated. This allows you to define parameterized SOM class templates. You may specify multiple –u options on the same command line. If the *name* specified is the same as a user–defined symbol name already defined in the **somtempl** configuration file, the substitution *value* on the command line takes precedence.

–S

This option inhibits substitution of template symbols. The files generated by **somtempl** will contain the content of the template file without substitution. This may be useful for creating your own additions to the template file.

Modifying and adding templates to somtempl

You may add templates or modify existing templates by making changes to **somtempl**'s configuration and template files, somtempl.cfg and somtempl.app. This section describes the contents of these files in more detail and explains how to modify these files.

Configuration file

The somtempl.cfg file defines the templates that can be generated by **somtempl**. There are two sections in this file that can be modified by users. The first section is for user–defined symbols. (Comments in the somtempl.cfg file indicate where the user–defined symbols should be placed.) A user–defined symbol is defined with the syntax:

```
symbolname=value
```

An example of such a symbol, shown below, can be found in the somtempl.cfg file.

```
USERDEFINEDTYPE=long
```

The string "USERDEFINEDTYPE" within a template file (somtempl.app) will be replaced by the string value "long" when the file containing the symbol is generated. The value associated with a user–defined symbol can be modified when **somtempl** is run, by using the –u option. For example, the above definition could be changed by adding the option:

```
–u USERDEFINEDTYPE=string
```

The second section that can be modified in a configuration file is the portion that defines the files to be generated for a template. The syntax of this portion of the configuration file is shown below:

```
:<template_name>|<template_title>|templatefile==<filename>
   <template_filename> [template_section_name]
   ...
[>[?<AIX|OS2|W16>]]
```

template_name

The template name is a single string that uniquely identifies the template.

template_title

The title can be multiple strings. The title, along with the template's name, is shown when the **somtempl** help is displayed.

filename

This is the name of the template file that contains the aggregate contents of the files named by *template_filename*. It can be any text file, although it is typically somtempl.app.

template_filename

This is the name of a file that will be generated by **somtempl**. You include one *template_filename* entry for each file you want **somtempl** to generate. The specified file name may contain the string __SOM_STEM__. __SOM_STEM__ will be replaced by the file stem specified with the –s command line option. When the –s option is not used, __SOM_STEM__ is replaced by xxxx. The generated contents of *template_filename* are located via a label of the form

```
:<template_name>_<template_filename>
```

within the templatefile. If a *template_section_name* is given along with a *template_filename*, the generated contents of *template_filename* are located via a label of the form:

```
:<template_section_name>
```

The *template_section_name* option allows you to define a single label that can be used by multiple templates.

>

This character, placed in the first column, directs the strings that follow on that line to standard error output – usually the display – after the files named by *template_filename* have been generated. You can tailor your message by system by including an optional system name following the > character. Specifically, >?AIX outputs only on AIX systems, >?OS2 outputs only on OS/2 systems and >?W16 outputs only on Windows systems.

#

This character, placed in the first column, indicates the line is a comment.

A complete example for the template named "idl_dll", contained in the supplied somtempl.cfg, is shown below:

```
# Template: idl_dll
# Title: Basic SOM Class Library
# Contents of files in: somtempl.app
:idl_dll|Basic SOM Class Library|templatefile==somtempl.app
# Generate the following 8 files.
# Note: the __SOM_STEM__.idl file is just the
#       generic IDL file.
    Makefile
    makefile.32
    makefile.w16
    __SOM_STEM__.def
    __SOM_STEM__.exp
    __SOM_STEM__.idl gidl
    __SOM_STEM__init.c
    __SOM_STEM__w16.def
# Output the following after the files are generated.
>    Application __SOM_CLASS_NAME__ generation completed.
>?AIX    Please type:
>?AIX        make
>?AIX    to build your __SOM_STEM__.dll.
>?OS2    Please type:
>?OS2        nmake -f makefile.32
>?OS2    to build your __SOM_STEM__.dll.
>?W16    Please type:
>?W16        nmake -f makefile.w16
>?W16    to build your __SOM_STEM__.dll.
```

Template file

A template file contains the aggregate contents of the files that have been defined within the **somtempl** configuration file. A template file contains the contents of multiple files to be generated. Each file generated is located located within a template file via a label. As described in the previous section, the label name may be either:

```
:<template_name>_<template_filename>
```

or

```
:<template_section_name>
```

The lines defining the generated file contents follow the label, up to but not including the line containing the next label.

Template file symbols

Within a template file, there are several symbols that can be used to allow a template to be generated based on the command line options used when **somtempl** is run. The value for each symbol is determined when **somtempl** is run; that is, each value is substituted in place of the corresponding symbol when files are generated. These symbols are in addition to the user–defined symbols that can be defined in the **somtempl** configuration file. The available symbols are described below: (Review the contents of the default template file somtempl.app to see in more detail how these symbols are used.)

Symbol

__SOM_CLASS_NAME__

This symbol is replaced by the name of the class specified with the –n option or with the default **defaultClass**.

__SOM_META_NAMES__

This symbol is used within IDL files and is replaced by the IDL file to be included for the metaclass named with the –m option. This defaults to **#include <somcls.idl>**.

__SOM_PARENT_NAMES__
: This symbol is replaced by the names of the parent classes specified with the –p option. This defaults to **SOMObject**. When more than one parent class is specified, the parent class names of the value of this symbol will be separated by commas.

__SOM_STEM__
: This symbol can be used in file names or wherever a unique name is needed. It corresponds to the stem specified with the –s command line option. It defaults to **xxxx**.

__SOM_OBJS__
: This symbol is used in make files. It names the object files that the generated application will need to build. Typically, the value of this symbol is xxxx.o. When parent class source is also generated, this object file name is also included in the value of this symbol. For example: parent.o xxxx.o. A metaclass object file name may also be added if generated.

__SOM_OBJS_RESPONSE__
: This symbol is used in linker response files and is similar to __SOM_OBJS__ except that file names are separated with a "+" character.

__SOM_OBJS_C_DEPS__
: This symbol is used in make files. It includes a line for each dependency to build an object file from a C source file. For example: xxxx.o: xxxx.ih xxxx.c. The dependencies for parent and metaclasses are also included in the symbol if they are generated.

__SOM_OBJS_CPP_DEPS__
: This symbol is used in make files. It includes a line for each dependency to build an object file from a C++ source file. For example: xxxx.o: xxxx.xih xxxx.cpp. The dependencies for parent and metaclasses are also included in the symbol if they are generated.

__SOM_INIT_BODY__
: This symbol is used in the body of a SOMInitModule procedure. It contains calls to the NewClass procedure of the classes generated. For example:

```
parentNewClass(parent_MajorVersion, parent_MinorVersion);
defaultClassNewClass(defaultClass_MajorVersion,
                     defaultClass_MinorVersion);
```

__SOM_C_HEADER_FILES__
: This symbol is used in C source code to include the generated class header files. For example:

```
#include "pppp.h"
#include "xxxx.h"
```

__SOM_C_HEADER_DELETE_FILES__
: This symbol is used in make files. It names the C header files to be deleted when the directory is cleaned of unnecessary files. For example:

```
-$(DELETE) pppp.h
-$(DELETE) xxxx.h
```

__SOM_CPP_HEADER_FILES__
: This symbol is used in C++ source code to include the generated class header files. For example:

```
#include "pppp.xh"
#include "xxxx.xh"
```

__SOM_CPP_HEADER_DELETE_FILES__
This symbol is used in make files. It names the C++ header files to be deleted when the directory is cleaned of unnecessary files. For example:

```
-$(DELETE) pppp.xh
-$(DELETE) xxxx.xh
```

__SOM_EXPORT_C_NAMES__
This symbol is used in module definition files to name external entry points for the generated classes. For example:

```
_parentCClassData
_parentClassData
_parentNewClass
_defaultClassCClassData
_defaultClassClassData
_defaultClassNewClass
```

__SOM_EXPORT_PASCAL_NAMES__
This symbol is used in module definition files to name external entry points for the generated classes. For example:

```
parentCClassData
parentClassData
parentNewClass
defaultClassCClassData
defaultClassClassData
defaultClassNewClass
```

__SOM_META_CLASS_NAME__
This symbol is replaced by the class name of the metaclass specified with the –m option.

__SOM_PARENT_CLASS_NAME__
This symbol is replaced by the class name of the parent specified with the –p option. When more than one –p option is specified, this symbol represents the first parent named.

__SOM_DLLSTEM__
This symbol is used to name overall output file names, such as a DLL or library produced by a make file. It defaults to **xxxx** and can be set via the –s option. The value of this symbol is constant, even when parent and metaclass files are generated (that is, where __SOM_STEM__ is determined by the –p and –m options.)

Adding templates

You can add new templates to the somtempl.cfg and somtempl.app files with any text editor. You may find that this is the most convenient approach. There is, however, an alternate way to add templates to these files that may simplify your task – you can use the **–g** option. The following are the recommend steps to use the –g option:

1. Make backup copies of the current somtempl.cfg and somtempl.app files.
2. Generate an existing template with the –S option. This option generates a template's files without doing any symbol substitution. For example, first generate the idl_dll template:

```
somtempl -a idl_dll -S
```

This will give you a set of files that is ready to be placed back into the template file somtempl.app as a new template.

3. Next, modify the set of generated template files to include the changes you want in your new template.
4. Use the –g and –T options with the names of the generated template files to add a new template to the configuration and template files. For example:

```
somtempl –T ”This is my modified template” –g newone Makefile
   makefile.32 makefile.w16 xxxx.def xxxx.exp xxxx.idl xxxxinit.c
   xxxxw16.def
```

   This will add a section in the somtempl.cfg file for the template named “newone” with a title of “This is my modified template”. The named files (Makefile, makefile.32, ...) will be added to somtempl.app.

5. Finally, edit the somtempl.cfg and somtempl.app files to make any finishing touches to your new template. For example, add text to the configuration file that should be displayed when the template is generated:

```
>?AIX    Please type:
>?AIX        make
>?AIX    to build your __SOM_STEM__.dll.
>?OS2    Please type:
>?OS2        nmake –f makefile.32
>?OS2    to build your __SOM_STEM__.dll.
>?W16    Please type:
>?W16        nmake –f makefile.w16
>?W16    to build your __SOM_STEM__.dll.
```

Note: When you use the –g option, the only reverse symbol substitution that is done is for the stem name (–s) and class name (–n). That is, the names of the files in the example above are stored in the configuration and template files using __SOM_STEM__ instead of xxxx. Likewise, defaultClass is replaced with __SOM_CLASS_NAME__.