somFree Compiler and Emitter Framework

User's Guide

Introduction

somFree Compiler and Emitter Framework is a free open source binary compatible reimplementation of IBM SOM Compiler and Emitter Framework. It is tries to be as compatible as possible on API and ABI level.

Changes

Changes from original somFree compiler:

- Most of internal structures now also present as in old IBM SOM 2.1 NT Toolkit.
- New emitters:
  - LNK - Open Watcom WLink support.
  - DUMP - displays structures, available to emitter.
  - PAS - Pascal client support.
  - IPAS - Pascal implementation classes support.
- SOM Compiler library now mostly documented.
- SOMLINK style functions for most of SOM Compiler library added.
- Emitters now IBM SOM 2.1 and IBM SOM 3.0 compatible without recompilation.
- somFree Compiler supports IBM SOM 2.1, IBM SOM 3.0, and somFree 1.0 emitters.
- somtShowEntry function outputs more info.
- Undocumented SOMTTypes now documented.
- SOMIPC now supports IDL 4.2 specification.
- CORBA C Language Mapping Specification 1.0 now supported by default instead of SOM C Language mapping.
- Added support of OIDL files

somFree Compiler

The somFree Compiler is a tool to produce various file formats from Interface Definition Language (IDL) files or Object Interface Definition Language (OIDL) files. somFree Compiler reads IDL or OIDL file and produces an abstract graph tree. Using abstract tree, somFree Compiler generates an object graph tree. After the object graph is ready, somFree Compiler produces an output using template.

The somFree Compiler uses DLL-name based loading of classes libraries (other programs can user another approach, like WPS does. WPS uses an Interface Repository to find corresponding class). Most of the somFree Compiler classes libraries it is implementation of corresponding emitter. Emitters can be created with help of Emitter Framework.

somFree Compiler actually is a client program which uses Emitter Framework classes. somFree Compiler is open source program with an open architecture. The only things that couldn't be easily extended are parser, abstract graph builder and object graph builder. Other things can be shadowed and replaced by our own.

Let's look at somFree Compiler command line syntax to understand how to produce corresponding skeleton code from somFree Compiler template (below is somFree Compiler help screen):

```
sc [-C:D:E:I:S:VU:cd:hi:m:prsvw] f1 f2 ...
Where:
      -C <n>            - size of comment buffer (default: 200000)
      -D <DEFINE>       - same as -D option for cpp.
      -E <var>=<value>  - set environment variable.
      -I <INCLUDE>      - same as -I option for cpp.
      -S <n>            - size of string buffer (default: 200000)
      -U <UNDEFINE>     - same as -U option for cpp.
      -V                - show version number of compiler.
      -c                - ignore all comments.
      -d <dir>          - output directory for each emitted file.
      -h                - this message.
      -i <file>         - use this file name as supplied.
      -m <name[=value]> - add global modifier.
      -p                - shorthand for -D__PRIVATE__.
      -r                - check releaseorder entries exist (default: FALSE).
      -s <string>       - replace SMEMIT variable with <string>
      -u                - update interface repository.
      -v                - verbose debugging mode (default: FALSE).
      -w                - don't display warnings (default: FALSE).

Modifiers:
      addprefixes : adds `functionprefix' to method names in template file
      [no]addstar : [no]add `*' to C bindings for interface references.
            corba : check the source for CORBA compliance.
              csc : force running of OIDL compiler.
       emitappend : append the emitted files at the end of the existing
file.
         noheader : don't add a header to the emitted file.
            noint : don't warn about "int" causing portability problems.
           nolock : don't lock the IR during update.
             nopp : don't run the source through the pre-processor.
             notc : don't use typecodes for emit information.
       nouseshort : don't generate short names for types.
        pp=<path> : specify a local pre-processor to use.
         tcconsts : generate CORBA TypeCode constants.

Note: All command-line modifiers can be set in the environment
by changing them to UPPERCASE and preappending "SM" to them.

Environment Variables:
      SMEMIT=[h;ih;c;xh;xih;xc;def;ir;pdl]
        : emitters to run (default : h;ih).
```

```
      SMINCLUDE=<dir1>[;<dir2>]+
        : where to search for .idl and .efw files.
      SMKNOWNEXTS=ext[;ext]+
        : add headers to user written emitters.
      SMTMP=<dir>
        : directory to hold intermediate files.
      SOMIR=<path>[;<path>]+
        : list of IRs to search.

Pragmas:
      #pragma somemittypes on          : turn on emission of global types.
      #pragma somemittypes off         : turn off emission of global types.
      #pragma modifier <modifier stm>; : instead of modifier statement.
```

Now let's explain some command line switches deeper.

First of the most interesting switch is -s. By default somFree Compiler uses SMEMIT environment variable to determine which emitter to use. Look at emit*.dll files for corresponding emitter. Using switch -s you can change default logic and select one-time emitter instead of global emitters. In easy situation you need only one emitter (say, C emitter). In complex situations you need use more emitters (say, C, H, DEF and IH emitters). You can create your own emitter to produces, for example, some sort of documentation and other stuff.

Another interesting switch is -m. Using -m you can set and/or unset so named modifiers. Modifiers allow you to change default behaviour of emitter and compiler. As example, by default compiler adds new methods or modifies existent. You can tell compiler just add new text to end of file. Modifiers can control emitters. addstart and noaddstar controls C emitter to add or not add pointer sign (*) to references of objects.

Switch -u adds or updates Interface Repository with new information about class interface. Interface repository filename controlled by SOMIR environment variable. This thing useful to add info for Object REXX access and other things which uses Interface Repository.

Other switches are like for standard C/C++ preprocessor and not described here.

Now let's play with somFree Compiler. Most often, you need to create interface files for C/C++ client programs. Usually you need to call the SOM Compiler as following:

```
sc -sdef somobj.idl
sc -sh somobj.idl
```

In case of C++ you need to call:

```
sc -sdef somobj.idl
sc -sxh somobj.idl
```

Of course, not very nice to call somFree Compiler so often. And somFree Compiler provides such functionality:

```
sc -sdef;h;xh somobj.idl
```

The above command will do exactly as all recent commands.

The above emitters were designed for IBM toolset. Nowadays, developers also use GCC or Open Watcom Compilers. The problem here is that Watcom Linker doesn't support .DEF files, but has its own .LNK linker files. In case of one or two classes no many problems to convert .DEF files to .LNK files manually. But such approach just ugly for MUCH classes. So, one of good solution is write REXX script for DEF→LNK conversion. But somFree Compilers contains Open Watcom Linker Emitter for such approaches.

Now let's talk about internals of somFree Compiler. somFree Compiler designed in the way as most of C compilers implemented. It is exists of following parts:

- IDL Preprocessor
- IDL Parser
- OIDL Preprocessor
- OIDL Parser
- Emitter Framework

somFree Compiler first calls IDL or OIDL Preprocessor. Output of IDL/OIDL Preprocessor goes to IDL/OIDL Parser. IDL/OIDL Parser creates Object tree from IDL/OIDL source. Object tree, using templates and emitters, stored to file.

As you see, most of the parts can be extended or replaced by its own implementation. For example, we can reuse CPP instead of SPP. Why not? Just support required command-line switches for compatibility. Also, default emitters can be rewritten. For C and C++ emitter it is not so hard. For other, more structured languages, like Pascal, Modula, etc. emitter creation is more hard work, but it is also possible.

Actually, we can extend and rewrite somFree Compiler as we want.

Usage of somFree Compiler will not be problem for most of you. But understanding some details about compiler internals makes life easer.

SOM Interface Definition Language

Latest IBM SOM 3.0 supports CORBA IDL mostly at level of CORBA 1.1. somFree supports CORBA IDL 4.2 with all extensions found in SOM IDL.

- Include Directives (optional)
- Type and Constant Declarations (optional)
- Exception Declarations (optional)
- Interface Declarations (optional)
- Module declaration (optional)

Let's try to define our class interface.

Interface Definition Language (IDL) is the core of System Object Model. All classes have definition of its interface via IDL. With help of somFree Compiler IDL file can be translated to various formats, including various language bindings. For example, to produce C header you can run

```
sc -s"h" somcls.idl
```

to produce DEF file you can run

```
sc -s"def" somcls.idl
```

somFree Compiler uses emitter to produce corresponding language binding. You can create new bindings emitter using Emitter Framework.

First of all, think about your class. What it must do? Define them in terms of object. Propose attributes and methods of class.

Ok. Imagine, we need class to have access to Java objects. Let's write a class interface in terms of Interface Definition Language.

```
#include <somobj.idl>

interface JavaObject : [[SOMObject]]
{
  implementation
  {
    somDefaultInit: override;
    // Init Java Virtual Machine (if no current) and create Java object
    somDefaultDestruct: override;
    // Destruct Java object and close Java Virtual Machine (if needed)
  }
}
```

As you can see, no many problems. Syntax of IDL too closest to C-like languages. First thing you need is to include definitions of parent classes. In our case it is '`SOMObject`' definition. Generic Java object doesn't need to have any methods. Only thing '`JavaObject`' will do its check for existence of Java Virtual Machine and execution of it if required.

On object destruction checks is Java Virtual Machine still required will be done and it will be destroyed if not required. Also same constructor and destructor will call corresponding constructor and destructor of Java object.

Classic IDL file contains definitions like

```
interface <class> : <parent_class>
{
  attribute <type> <name>

  <type> <method>(<parameters>)
}
```

[http://www.omg.org OMG] IDL doesn't support methods override. SOM IDL has such feature (and incompatibility with OMG CORBA). This is done via keyword '`implementation`'. To solve problems with other IDL compilers such part must be wrapped to #ifdef structure:

```
#include <somobj.idl>
```

```
interface JavaObject : [[SOMObject]]
{

  #ifdef __SOMIDL__
  implementation
  {
    somDefaultInit: override;
    // Init Java Virtual Machine (if no current) and create Java object
    somDefaultDestruct: override;
    // Destruct Java object and close Java Virtual Machine (if needed)
  }
  #endif
}
```

Such approach well known in C-world, but also have some problems. For example, IDL of Document Object Model (DOM) (Yes, [http://www.w3.org W3C] DOM uses same IDL as SOM and CORBA) has attribute '`implementation`'. As result, somFree Compiler has some problems with IDL compilation.

As a first step we'll create SOM class interface for java.lang.Object. It can be done with help of javah tool.

```
javah -jni java.lang.Object
```

As result you'll have such file:

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class java_lang_Object */

#ifndef _Included_java_lang_Object
#define _Included_java_lang_Object
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     java_lang_Object
 * Method:    hashCode
 * Signature: ()I
 */
JNIEXPORT jint JNICALL Java_java_lang_Object_hashCode
  (JNIEnv *, jobject);

/*
 * Class:     java_lang_Object
 * Method:    notify
 * Signature: ()V
 */
JNIEXPORT void JNICALL Java_java_lang_Object_notify
  (JNIEnv *, jobject);

/*
```

```
 * Class:     java_lang_Object
 * Method:    notifyAll
 * Signature: ()V
 */
JNIEXPORT void JNICALL Java_java_lang_Object_notifyAll
  (JNIEnv *, jobject);

/*
 * Class:     java_lang_Object
 * Method:    registerNatives
 * Signature: ()V
 */
JNIEXPORT void JNICALL Java_java_lang_Object_registerNatives
  (JNIEnv *, jclass);

/*
 * Class:     java_lang_Object
 * Method:    wait
 * Signature: (J)V
 */
JNIEXPORT void JNICALL Java_java_lang_Object_wait
  (JNIEnv *, jobject, jlong);

/*
 * Class:     java_lang_Object
 * Method:    getClass
 * Signature: ()Ljava/lang/Class;
 */
JNIEXPORT jclass JNICALL Java_java_lang_Object_getClass
  (JNIEnv *, jobject);

/*
 * Class:     java_lang_Object
 * Method:    clone
 * Signature: ()Ljava/lang/Object;
 */
JNIEXPORT jobject JNICALL Java_java_lang_Object_clone
  (JNIEnv *, jobject);

#ifdef __cplusplus
}
#endif
#endif
```

So, this header can be used to generate actual class interface using this script:

```
/* REXX - our best dog */

do while lines('java_lang_Object.h')
  s=linein('java_lang_Object.h');
  parse value s with x 'Header for class' name '*/'
```

```
   if name\='' then
   do
     classname=strip(name)
     say '#include <JavaObject.idl>'
     say ''
     say 'interface '||classname||' : JavaObject'
     say '{'
   end
   parse value s with x 'Method:' name
   if name\='' then
   do
     /* skip 3 lines */
     s=linein('java_lang_Object.h');
     s=linein('java_lang_Object.h');
     s=linein('java_lang_Object.h');
     s2=linein('java_lang_Object.h');
     interpret("parse value s with 'JNIEXPORT' type 'JNICALL
Java_"||classname||"_' name")
     name=strip(name)
     s2=strip(s2)
     parse value s2 with start 'JNIEnv *, jobject' end
     s2=start||end
     parse value s2 with x ', ' y
     if x='(' then s2='('||y
     say '  '||type||name||' '||s2
   end
 end
 say '}'
```

As result, you'll have following:

```
 #include <JavaObject.idl>

 interface java_lang_Object : JavaObject
 {
    jint hashCode ();
    void notify ();
    void notifyAll ();
    void registerNatives (JNIEnv *, jclass);
    void wait (jlong);
    jclass getClass ();
    jobject clone ();
 }
```

Using such approach you can easily make SOM wrappers for all Java classes. Using Java_JNI_API you can create Java classes and using SOM wrappers you integrate Java code to SOM-based applications. Because SOM API more generic then Java API you can use any available language bindings for development. Also, you can start to extend Java classes by native code with help of SOM engine.

SOM Object Interface Definition Language

SOM Object Interface Definition Language is a pre-IDL object definition language used before IBM SOM 2. Since IBM SOM 2 uses CORBA IDL as defined in OMG CORBA 1.1. SOM Object Interface Definition Language (OIDL) is a simple definition language and not recommended to use. SOM Compiler support is only for compatibility with old source code. OIDL support implementation mostly based on [2] and various OIDL source files found on the Web. OIDL consist of sections set:

- Include section (optional)
- Class section (required)
- Release order section (optional)
- Parent class section (required)
- Passthru section (optional)
- Metaclass section (optional)
- Data section (optional)
- Methods section (optional)

Include section

Include section is optional and contains names of OIDL files with definition of parent class, metaclasses and private interfaces of ancestor classes.

```
[#include ( <ancestor> | "ancestor" ) ] *
#include ( <parent> | "parent")
[#include ( <metaclass> | "metaclass" ) ]
```

ancestor is the name of the OIDL file containing the private part of an ancestor class' interface needed in the definition of this class. If ancestor is enclosed in angle brackets (<>), the search for the file will begin in system-specific locations. If parent is enclosed in double quotation marks (""), the search for the file will begin in the local context, then move to the system-specific locations.

parent is the name of the OIDL file containing the parent class of the class for which the Include statement is provided. If parent is enclosed in angle brackets (<>), the search for the file will begin in system-specific locations. If parent is enclosed in double quotation marks (""), the search for the file will begin in the local context, then move to the system-specific locations.

metaclass is the OIDL file containing the metaclass of the class for which the include statement is provided. If metaclass is enclosed in angle brackets (<>), the search for the file will begin in system-specific locations. If metaclass is enclosed in double quotation marks (""), the search for the file will begin in the local context, then move to the system-specific locations.

Class section

```
class: name
  [, file stem = stem]
  [, external stem = stem]
  [, function prefix = prefix |
  , external prefix = prefix |
```

```
  , classprefix = prefix]
  [, major version = number]
  [, minor version = number]
  [, global | local];
  [, classInit = function];
[description]
```

Release order section

```
release order: name [, name ]* ;
```

Parent class section

```
parent [class]: name;
description
```

Passthru section

```
[passthru: language.suffix, [ before | after ];
line 1
line 2
endpassthru; [description]]*
```

Metaclass section

```
metaclass: name;
[description]
```

Data section

```
data:
[description1 ]
[declaration [ , private | , public | , internal] [, class];
[ description2]]*
```

Methods section

```
methods:
[description1 ]
[[group: name;
[ description2]]
[method prototype
[, public | , private]
```

```
[, method | , procedure]
[, class]
[, offset | , name lookup]
[, local | , external]
[, use = name];
[descriptionS]]*
[override: method name
[, public | , private]
[, class]
[, local | , external]
[, use = name];
[description4]] *
```

Programmer's Guide

Introduction

somFree compiler is a tool to convert various interface definition languages to another one or language bindings. somFree compiler frontend is a sc or somc command which control workflow. Because somFree compiler and Emitter Framework modeled after IBM SOM Compiler from here SOM Compiler term will be used. Most of somFree Compiler and Emitter Framework and SOM Compiler and Emitter Framework are same and binary compatible at the documented level. Internal structures of somFree and IBM versions are different.

Structure of SOM Compiler and Emitter Framwork

SOM Compiler at file level consist of:

- SOM Compiler frontend
  - sc [Linux]
  - sc.exe [OS/2, Windows]
  - somc.exe [Windows]
- IDL SOM Pre-processor
  - somcpp [Linux]
  - somcpp.exe [OS/2, Windows]
- IDL SOM Compiler
  - somipc [Linux]
  - somipc.exe [OS/2, Windows]
- OIDL SOM Pre-processor
  - spp [Linux]
  - spp.exe [OS/2, Windows]
- OIDL SOM Compiler
  - somopc [Linux]
  - somopc.exe [OS/2, Windows]
- SOM Compiler Library

    - somc.so [Linux]
    - somc.dll [OS/2, Windows]
- SOM Emitter Framework
    - some.so [Linux]
    - some.dll [OS/2, Windows]
- Emitters
    - emit*.so [Linux]
    - emit*.dll [OS/2, Windows]
- Public IDL files generator
    - pdl [Linux]
    - pdl.exe [OS/2, Windows]

Currently SOM Compiler provides following emitters:

- IDL - IDL Emitter
- CSC - OIDL Emitter
- SC - OIDL public emitter
- GEN - Generic Emitter
- IR - Interface Repository Emitter
- H - C Binding public header files
- C - C Binding implementation template file
- IH - C Binding implementation header files
- XH - C++ Binding public header files
- XIH - C++ Binding implementation header files
- DEF - DEF Module Definition file
- LNK - LNK Module Linking file
- HC
- IMOD - SOM Module initialization emitter
- MODS - List of class modifiers
- PDL - Private IDL emitter
- PH
- PSC - OIDL private emitter
- UC
- UXC
- XPH
- XTM
- PAS - Pascal client library for use of SOM
- IPAS - Pascal implementation library to write SOM classes.

Some of Emitters uses Templates such as:

- cpp.efw
- ctm.efw
- gen_c.efc
- gen_c.efs
- gen_c.efw
- gen_cpp.efw
- gen_def.efw
- gen_emit.efc
- gen_emit.efs
- gen_emit.efw
- gen_emit.efx

- gen_idl.efw
- gen_make.efc
- gen_make.efs
- gen_make.efw
- gen_make.efx
- gen_mk32.efc
- gen_mk32.efs
- gen_mk32.efw
- gen_mk32.efx
- gen_mknt.efs
- gen_mknt.efw
- gen_mknt.efx
- gen_nid.efw
- gen_temp.efw
- imod.efw

Interaction of SOM Compiler components

Emitter Framework is a set of classes and SOM Compiler tool. Emitter Framework is used to produce various file formats from the SOM Interface Definition Language files. Emitter Framework classes consist of Emitter classes and Entry classes. Classes can be shadowed. This means a programmer can replace original classes with his own classes. So the SOM Compiler can be highly customized. The only things hard-coded (and closed source) are the IDL file reader and abstract graph builder.

Before starting description of Emitter Framework let's talk about SOM Compiler. We already talked briefly about SOM Compiler. But for emitters we need to know internals of SOM Compiler much better.

Let's start from visible parts of SOM Compiler that requires for its work the following files:

- sc.exe, somc.dll and somc.msg - Main part of compiler.
- somcpp.exe - SOM Preprocessor
- somipc.exe - Goals not known. Seems just execute different emitters
- emit*.dll - Emitters
- *.efw - Emitter templates

`sc.exe` is general part of compiler. Let's try to investigate some internals of sc.exe. First of all we can switch on verbose output and look on it:

```
Running shell command:
somcpp -D__OS2__  -I. -IC:\os2tk45\h -IC:\os2tk45\idl -
IC:\os2tk45\som\include \
      -D__SOMIDL_VERSION_1__ -D__SOMIDL__ -C somobj.idl >
C:\var\temp\0a500000.CTN
somipc -mppfile=C:\var\temp\0a500000.CTN -v -e emith -e emitih -e emitctm -e
emitc \
      -o somobj somobj.idl
Loading  emith.
"SOMObject"
Unloading  emith.
Loading  emitih.
```

```
"SOMObject"
Unloading  emitih.
Loading  emitctm.
"SOMObject"
Unloading  emitctm.
Loading  emitc.
"SOMObject"
Unloading  emitc.
Removed "C:\var\temp\0a500000.CTN".
```

Not so many info, but some information here. If we look at SMINCLUDE environment variable:

```
SMINCLUDE=.;C:\os2tk45\h;C:\os2tk45\idl;C:\os2tk45\som\include;
```

then we will see all paths in -I option.

Considering -D is same as for CPP we can see three symbols defined: * OS2 * SOMIDL_VERSION_1 * SOMIDL somobj.idl it is file we emitted and CTN file is output from preprocessor. The only unknown switch is -C. After small playing we can see it means “leave comments”.

So, we can try to replace somcpp with some preprocessor. In [http://www.osfree.org osFree] project we tried to use [http://mcpp.sourceforge.net MCPP] preprocessor. Results is well.

sc.exe reads SMINCLUDE variable and puts its content to -I options of somcpp.exe and redirect output to temporary file.

Ok. Now we can try to detect what is somipc.exe. If we try to execute it with command line pointed above, then we will see: Loading emith. “SOMObject” Unloading emith. Loading emitih. “SOMObject” Unloading emitih. Loading emitctm. “SOMObject” Unloading emitctm. Loading emitc. “SOMObject” Unloading emitc.

Heh. Actually, somipc.exe is a real SOM Compiler. Not sc.exe. sc.exe only prepares the input file for the compiler and handles command line and environment variables.

After some playing we can see, somipc returns 0 if all ok and -1 if error.

So, somipc.exe parses preprocessed IDL file and builds Abstract graph. From Abstract graph Object Graph are build. After this somipc.exe calls one by one all emit*.dll files according to -e switches. emit*.dll are set of DLLs with SOM classes.

Drawing here!!

SOM Compiler IDL SOM Preprocessor IDL SOM Compiler Emitter Template

```
        OIDL SOM Preprocessor    OIDL SOM Compiler
```

Разрисовать по аналогии с со структурой, что в патентах и документации по SOM, но с учетом наличия OIDL и SOMC.

SOM Compiler sc or somc is a frontend which controls basic workflow. Depending on source file extension it call or IDL or OIDL pre-processor and, after preprocessing, IDL or OIDL compiler. IDL or OIDL compiler builds abstract syntax graph using Entry structure. Entry structure contains information

about entry type, pointer to object wrapper and all information about object specific attributes.

Note! Entry structure is not documented and differs in somFree and IBM SOM versions.

IDL or OIDL calls required emitters with root Entry structure on emitter entry. Emitter requests root object wrapper and, using or not using template faculty, process all graph using Object Syntax Graph. Object Syntax Graph generates required Entry objects on demand.

Emitter is a subclass of '`SOMTEmitC`' class. Emitter used to produce output file using template file from object graph of the SOM Interface Definition Language file. Physically emitter represented as DLL with name EMIT<identificator>.DLL. For C headers emith.dll emitter DLL is used. For C++ headers emitxh.dll emitter DLL is used. Emitter DLL contains only one entry with ordinal 1 and name '`emit`'.

```
SOMEXTERN FILE * SOMLINK emit(char *file, Entry * cls, Stab * stab);
```

'`emit`' function creates emitter object (from emitter class, which based on '`SOMTEmitC`') and calls '`somtGenerateSections`' method.

Usually an emitter file can be generated using '`newemit.cmd`' script (can be found at Hobbes in SOMObjects toolkit).

```
newemit -C <className> <file_stem>
```

To emitter passed Entry object which is root of Object Graph. The root object can be an interface or module class. Processing of such classes slightly different.

Last part of Emitter Framework is template files. Template files allow you to make some control of emitting process. Templates are usual text files with extension *.efw. Here you can modify output for your wish. Not all emitters support templates.

So, now you have some imagination about that Emitter Framework is and how it works.

Template faculty

Emitters uses template faculty to produce output file. Template file has structure divided by sections. Each section begins from section name ended by colon. Each emitter can use its own section names. Refer to corresponding emitter and Entry classes description for section names information. Here is template file example:

```
:copyrightS
This is example template
:templateS
/* Template output example */
<className>
```

Core of Template faculty is a Key-Value strings collection represented by SOMStringTableC class. All substitutable to template values stored in SOMStringTableC class instance. On template file process, First of all SOMTEmitC method somtSetPredefinedSymbols sets section names symbols. By default it is following sections:

prologSN	prologS
baseIncludesPrologSN	baseIncludesPrologS
baseIncludesSN	baseIncludesS
baseIncludesEpilogSN	baseIncludesEpilogS
metaIncludeSN	metaIncludeS
classSN	classS
metaSN	metaS
basePrologSN	basePrologS
baseSN	baseS
baseEpilogSN	baseEpilogS
constantPrologSN	constantPrologS
constantSN	constantS
constantEpilogSN	constantEpilogS
typedefPrologSN	typedefPrologS
typedefSN	typedefS
typedefEpilogSN	typedefEpilogS
structPrologSN	structPrologS
structSN	structS
structEpilogSN	structEpilogS
unionPrologSN	unionPrologS
unionSN	unionS
unionEpilogSN	unionEpilogS
enumPrologSN	enumPrologS
enumSN	enumS
enumEpilogSN	enumEpilogS
attributePrologSN	attributePrologS
attributeSN	attributeS
attributeEpilogSN	attributeEpilogS
interfacePrologSN	interfacePrologS
interfaceSN	interfaceS
interfaceEpilogSN	interfaceEpilogS
modulePrologSN	modulePrologS
moduleSN	moduleS
moduleEpilogSN	moduleEpilogS
passthruPrologSN	passthruPrologS
passthruSN	passthruS
passthruEpilogSN	passthruEpilogS
releaseSN	releaseS
dataPrologSN	dataPrologS
dataSN	dataS
dataEpilogSN	dataEpilogS
methodsPrologSN	methodsPrologS
methodsSN	methodsS
overrideMethodsSN	overrideMethodsS
overriddenMethodsSN	overriddenMethodsS
inheritedMethodsSN	inheritedMethodsS

methodsEpilogSN	methodsEpilogS
epilogSN	epilogS

Generic Emitter

Generic emitter is a generic template based emitter. It uses simplest template with only one section “template”. Main goal of Generic Emitter is to produce Generic framework emitter files. It is used by newemit tool to produce full set of files required to build new emitter. Добавить описание символов шаблона и описание, какой шаблон за что отвечает.

DEF Emitter

DEF emitter used to generate definition file for DLL creation using MS LINK. somFree version of emitter uses template file to generate DEF file. Original IBM SOM DEF Emitter uses hard coded generation. Добавить описание символов шаблона.

LNK Emitter

LNK emitter used to generate linking file for DLL creation using Watcom WLINK. somFree version of emitter uses template file to generate LNK file. Original IBM SOM DEF Emitter doesn't have such emitter. Добавить описание символов шаблона.

CSC, PSC, SC Emitters

CSC emitter used to generate OIDL class definition file (CSC) used in IBM SOM 1.0. somFree version of emitter uses template file to generate CSC file. Original IBM SOM CSC Emitter uses hard coded generation. Добавить описание символов шаблона.

IDL, PDL Emitters

IDL emitter used to generate IDL class definition file used in IBM SOM 2.0 and higher. somFree version of emitter uses template file to generate IDL file. Original IBM SOM IDL Emitter uses hard coded generation. Добавить описание символов шаблона.

Developing new emitter

somFree Emitter Framework provides templates and libraries for developing emitters compatible with both IBM SOM 2.1 and IBM SOM 3.0 compilers. Because of different ABI (refer Appendix 1 for more information) somFree emitters automatically configures for corresponding API.

CORBA C Language mapping

somFree Compiler support CORBA C Language Mapping Specification 1.0 [1]. CORBA C Language mapping slightly differ from SOM C Language mapping, used by original IBM SOM 2.1. CORBA C Language mapping is default for somFree Compiler. This chapter provides short description of mapping. For full description refer to [1].

SOM C Language mapping

SOM C Language mapping is a IBM SOM mapping variant. For some reason (most probably because variable arguments support) IBM SOM not exactly implements C Language Mapping Specification.

Programmer's reference

SOM Runtime C library

SOM Runtime C library somwm35i is a subset of C runtime library functions found to be used by IBM SOM 3.0 for NT emitters. SOM Runtime C library provided only for support of IBM SOM 3.0 for NT emitters. This is not full featured C library but compatibility layer and must not be used for development. Functions utilize IBM Optlink calling convention. This library required only under Windows NT systems.

List of emulated function and variables.

- _CRT_init
- _CRT_term
- _abort_in_progress
- _exception_dllinit
- _matherr
- fclose
- _fprintfieee
- strlen
- _sprintfieee
- strcmp
- strstr
- _ctype
- feof
- fgetc
- fgets
- fputs
- fread
- fseek
- fwrite
- memmove

- memset
- remove
- rename
- rewind
- strchr
- strcpy
- strlen
- strncmp
- strncpy
- strrchr
- strtok
- tolower
- memcpy
- strcat
- getenv
- _printfieee
- _sscanfieee
- exit
- stderr
- _putenv
- _terminate
- _PrintErrMsg
- _SysFindFirst
- _SysFindNext
- _SysFindClose
- malloc
- free
- strdup
- strpbrk

somFree Compiler library

somFree Compiler library somc is a set of helper functions for compiler tasks. Used by IBM SOM emitters. Library provided solely to provide support of IBM emitters. Must not be used to write new code.

somtfexists, somtfexistsSL function

```
SOMEXTERN BOOL somtfexists(char *file);
SOMEXTERN BOOL SOMLINK somtfexistsSL(char *file);
```

Check is file exists in paths.

Note: somtfexists version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtsearchFile, somtsearchFileSL function

```
SOMEXTERN char * somtsearchFile(char *file, char *fullpath, char *env);
SOMEXTERN char * SOMLINK somtsearchFileSL(char *file, char *fullpath, char
*env);
```

Search path using file and env dirs and return full path if exists.

Note: somtsearchFile version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somttraverseParents, somttraverseParentsSL function

```
SOMEXTERN int somttraverseParents(FILE *fp, Entry * cls, Entry *arg, int
(*fn)(FILE*,Entry*,Entry*), SMTraverse flg);
SOMEXTERN int SOMLINK somttraverseParentsSL(FILE *fp, Entry * cls, Entry
*arg, int (*fn)(FILE*,Entry*,Entry*), SMTraverse flg);
```

Note: somttraverseParents version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtloadSL function

```
SOMEXTERN EmitFn  SOMLINK somtloadSL(char *fileName, char *functionName,
void **modHandle);
```

Load emitter <fileName> and return pointer <EmitFn> to emit or emitSL function <functionName> and return handle <modHandle> of loaded module. This function switches somc to IBM SOM 3.0 ABI if emitSL function found or to IBM SOM 2.1 ABI if emit function found.

somtfindBaseEp, somtfindBaseEpSL function

```
SOMEXTERN Entry * somtfindBaseEp(Entry *ep);
SOMEXTERN Entry * SOMLINK somtfindBaseEpSL(Entry *ep);
```

Note: somtfindBaseEp version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtgetType, somtgetTypeSL function

```
SOMEXTERN Entry * somtgetType(char *name, SOMTTypes type);
SOMEXTERN Entry * SOMLINK somtgetTypeSL(char *name, SOMTTypes type);
```

Note: somtGetType version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtokfopen, somtokfopenSL function

```
SOMEXTERN FILE * somtokfopen(char *path, char *mode);
SOMEXTERN FILE * SOMLINK somtokfopenSL(char *path, char *mode);
```

Same as C fopen function.

Note: somtokfopen version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtokrename, somtokrenameSL function

```
SOMEXTERN int somtokrename(const char*, const char *);
SOMEXTERN int SOMLINK somtokrenameSL(const char*, const char *);
```

Note: somtokrename version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtopenEmitFile, somtopenEmitFileSL function

```
SOMEXTERN FILE * somtopenEmitFile(char *file, char *ext);
SOMEXTERN FILE * SOMLINK somtopenEmitFileSL(char *file, char *ext);
```

Note: somtopenEmitFile version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtisDbcs, somtisDbcsSL function

```
SOMEXTERN BOOL somtisDbcs(int c);
SOMEXTERN BOOL SOMLINK somtisDbcsSL(int c);
```

Note: somtisDbcs version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtremoveExt, somtremoveExtSL function

```
SOMEXTERN boolean somtremoveExt(char *name, char *ext, char *buf);
SOMEXTERN boolean SOMLINK somtremoveExt(char *name, char *ext, char *buf);
```

Remove extension from<name> and return to <buf>

Note: somtremoveExt version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtaddExt, somtaddExtSL function

```
SOMEXTERN char * somtaddExt(char *name, char *ext, char *buf);
SOMEXTERN char * SOMLINK somtaddExtSL(char *name, char *ext, char *buf);
```

Add <ext> extension to <name> filestem and return result in <buf>

Note: somtaddExt version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtarrayToPtr, somtarrayToPtrSL function

```
SOMEXTERN char * somtarrayToPtr(Entry *ep, char *stars, char *buf);
SOMEXTERN char * SOMLINK somtarrayToPtrSL(Entry *ep, char *stars, char
*buf);
```

Note: somtarrayToPtr version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtattNormalise, somtattNormaliseSL function

```
SOMEXTERN char * somtattNormalise(char *name, char *buf);
SOMEXTERN char * SOMLINK somtattNormaliseSL(char *name, char *buf);
```

Note: somtattNormalise version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtbasename, somtbasenameSL function

```
SOMEXTERN char * somtbasenameSL(char *path);
SOMEXTERN char * SOMLINK somtbasenameSL(char *path);
```

Return filename without path.

Note: somtbasename version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtctos, somtctosSL function

```
SOMEXTERN char * somtctos(Const *con, char *buf);
SOMEXTERN char * SOMLINK somtctosSL(Const *con, char *buf);
```

Note: somtctos version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtdbcsPostincr, somtdbcsPostincrSL function

```
SOMEXTERN char * somtdbcsPostincr(char **p);
SOMEXTERN char * SOMLINK somtdbcsPostincrSL(char **p);
```

Note: somtdbcsPostincr version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtdbcsPreincr, somtdbcsPreincrSL function

```
SOMEXTERN char * somtdbcsPreincr(char **p);
SOMEXTERN char * SOMLINK somtdbcsPreincrSL(char **p);
```

Note: somtdbcsPreincr version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtdbcsStrchr, somtdbcsStrchrSL function

```
SOMEXTERN char * somtdbcsStrchr(char *s, int c);
SOMEXTERN char * SOMLINK somtdbcsStrchrSL(char *s, int c);
```

Note: somtdbcsStrchr version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtdbcsStrrchr, somtdbcsStrrchrsL function

```
SOMEXTERN char * somtdbcsStrrchr(char *s, int c);
SOMEXTERN char * SOMLINK somtdbcsStrrchrSL(char *s, int c);
```

Note: somtdbcsStrrchr version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtdbcsStrstr, somtdbcsStrstrSL function

```
SOMEXTERN char * somtdbcsStrstr(char *s1, char *s2);
SOMEXTERN char * SOMLINK somtdbcsStrstrSL(char *s1, char *s2);
```

Note: somtdbcsStrstr version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somteptotype, somteptotypeSL function

```
SOMEXTERN char * somteptotype(Entry *ep, char *ptrs, char *buf);
```

```
SOMEXTERN char * SOMLINK somteptotypeSL(Entry *ep, char *ptrs, char *buf);
```

Note: somteptotype version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtgetDesc, somtgetDescSL function

```
SOMEXTERN char * somtgetDesc(Stab *stab, Entry *cls, Entry *method, char
*desc, BOOL addQuotes, BOOL use, BOOL versflg);
SOMEXTERN char * SOMLINK somtgetDescSL(Stab *stab, Entry *cls, Entry
*method, char *desc, BOOL addQuotes, BOOL use, BOOL versflg);
```

Note: somtgetDesc version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtgetVersion, somtgetVersionSL function

```
SOMEXTERN char * somtgetVersion(char *sccsid, char *version);
SOMEXTERN char * SOMLINK somtgetVersionSL(char *sccsid, char *version);
```

Note: somtgetVersion version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtgetgatt, somtgetgattSL function

```
SOMEXTERN char * somtgetgatt(char *s);
SOMEXTERN char * SOMLINK somtgetgattSL(char *s);
```

Note: somtgetgatt version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtnextword, somtnextwordSL function

```
SOMEXTERN char * somtnextword(const char *s, char *buf);
SOMEXTERN char * SOMLINK somtnextwordSL(const char *s, char *buf);
```

Note: somtnextword version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtnormaliseDesc, somtnormaliseDescSL function

```
SOMEXTERN char * somtnormaliseDesc(char *desc, char *normal);
SOMEXTERN char * SOMLINK somtnormaliseDescSL(char *desc, char *normal);
```

Note: somtnormaliseDesc version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtsatos, somtsatosSL function

```
SOMEXTERN char * somtsatos(char **sa, char *sep, char *buf);
SOMEXTERN char * SOMLINK somtsatosSL(char **sa, char *sep, char *buf);
```

Note: somtsatos version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtsearchFile, somtsearchFileSL function

```
SOMEXTERN char * somtsearchFile(char *file, char *path, char *envvar);
SOMEXTERN char * SOMLINK somtsearchFileSL(char *file, char *path, char
*envvar);
```

Note: somtsearchFile version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtskipws, somtskipwsSL function

```
SOMEXTERN char * somtskipws(const char *s);
SOMEXTERN char * SOMLINK somtskipwsSL(const char *s);
```

Note: somtskipws version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtstringFmt, somtstringFmtSL function

```
SOMEXTERN char * somtstringFmtSL(char *fmt, ...)
SOMEXTERN char * SOMLINK somtstringFmtSL(char *fmt, ...)
```

Allocate buffer for string, format it using <fmt> and return pointer to buffer.

Note: somtstringFmt version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somttype, somttypeSL function

```
SOMEXTERN char * somttype(SOMTType type);
SOMEXTERN char * SOMLINK somttypeSL(SOMTType type);
```

Return string representation of type of Entry structure except special case SOMTEmitterBeginE and SOMTEmitterEndE types.

Note: somttype version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

Warning: Depricated. Use somtEntryTypeName instead.

somtuniqFmt, somtuniqFmtSL function

```
SOMEXTERN char * somtuniqFmt(MemBuf *membuf, char *fmt, ...)
SOMEXTERN char * SOMLINK somtuniqFmtSL(MemBuf *membuf, char *fmt, ...)
```

Return unique formatted string.

Note: somtuniqFmt version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtargFlag, somtargFlagSL function

```
SOMEXTERN int somtargFlag(int *argc, char ***argv);
SOMEXTERN int SOMLINK somtargFlagSL(int *argc, char ***argv);
```

Note: somtargFlag version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtattjoin, somtattjoinSL function

```
SOMEXTERN int somtattjoin(register AttList *ap1, AttList *ap2);
SOMEXTERN int SOMLINK somtattjoinSL(register AttList *ap1, AttList *ap2);
```

Note: somtattjoin version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtdbcsLastChar, somtdbcsLastCharSL function

```
SOMEXTERN int somtdbcsLastChar(char *buf);
SOMEXTERN int SOMLINK somtdbcsLastCharSL(char *buf);
```

Note: somtdbcsLastChar version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtdbcsScan, somtdbcsScanSL function

```
SOMEXTERN int somtdbcsScan(char **buf);
SOMEXTERN int SOMLINK somtdbcsScanSL(char **buf);
```

Note: somtdbcsScan version uses default compiler calling convention. For IBM SOM 3.0 for NT it is

Optlink.

somtdiskFull, somtdiskFullSL function

```
SOMEXTERN int somtdiskFull(FILE *fp);
SOMEXTERN int SOMLINK somtdiskFullSL(FILE *fp);
```

Note: somtdiskFull version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtfclose, somtfcloseSL function

```
SOMEXTERN int somtfclose(FILE *fp);
SOMEXTERN int SOMLINK somtfcloseSL(FILE *fp);
```

Same as C fclose function.

Note: somtfclose version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtisparent, somtisparentSL function

```
SOMEXTERN int somtisparent(Entry *cls, Entry *parent);
SOMEXTERN int SOMLINK somtisparentSL(Entry *cls, Entry *parent);
```

Note: somtisparent version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtmget, somtmgetSL function

```
SOMEXTERN int somtmget(int setnum, int msgnum, char *msgbuf);
SOMEXTERN int SOMLINK somtmgetSL(int setnum, int msgnum, char *msgbuf);
```

Note: somtmget version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtmopen, somtmopenSL function

```
SOMEXTERN int somtmopen(char *filename);
SOMEXTERN int SOMLINK somtmopenSL(char *filename);
```

Note: somtmopen version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtmprintf, somtmprintfSL function

```
SOMEXTERN int somtmprintf(int setnum, int msgnum, ...);
SOMEXTERN int SOMLINK somtmprintfSL(int setnum, int msgnum, ...);
```

Note: somtmprintf version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtokremove, somtokremoveSL function

```
SOMEXTERN int somtokremove(char *file);
SOMEXTERN int SOMLINK somtokremoveSL(char *file);
```

Alias of C remove function.

Note: somtokremove version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtunload, somtunloadSL function

```
SOMEXTERN int somtunload(void *modHandle);
SOMEXTERN int SOMLINK somtunloadSL(void *modHandle);
```

Note: somtunload version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtwriteaccess, somtwriteaccessSL function

```
SOMEXTERN int somtwriteaccess(char *file);
SOMEXTERN int SOMLINK somtwriteaccessSL(char *file);
```

Note: somtwriteaccess version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtsmalloc, somtsmallocSL function

```
SOMEXTERN void * somtsmalloc(size_t nbytes, BYTE clear);
SOMEXTERN void * SOMLINK somtsmallocSL(size_t nbytes, BYTE clear);
```

Allocate <nbytes> of memory and fill it by zeroes if <clear> flag is set.

Note: somtsmalloc version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtaddGAtt, somtaddGAttSL function

```
SOMEXTERN void somtaddGAtt(MemBuf **membuf, AttList **ap, char *buf);
SOMEXTERN void SOMLINK somtaddGAttSL(MemBuf **membuf, AttList **ap, char
*buf);
```

Note: somtaddGAtt version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtcalcFileName, somtcalcFileNameSL function

```
SOMEXTERN void somtcalcFileName(char *def, char *over, char *ext);
SOMEXTERN void SOMLINK somtcalcFileNameSL(char *def, char *over, char *ext);
```

Note: somtcalcFileName version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtcleanFilesFatal, somtcleanFilesFatalSL function

```
SOMEXTERN void  somtcleanFilesFatal(int status);
SOMEXTERN void  SOMLINK somtcleanFilesFatalSL(int status);
```

Delete temporary files (if emitted file opened) and exit.

Note: somtcleanFilesFatal version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtemitTypes, somtemitTypesSL function

```
SOMEXTERN void somtemitTypes(FILE *fp, Mlist *mp, Stab *stab);
SOMEXTERN void SOMLINK somtemitTypesSL(FILE *fp, Mlist *mp, Stab *stab);
```

Note: somtemitTypes version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somterror, somterrorSL function

```
SOMEXTERN void somterror(char *file, long lineno, char *fmt, ...);
SOMEXTERN void SOMLINK somterrorSL(char *file, long lineno, char *fmt, ...);
```

Note: somterror version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtfatal, somtfatalSL function

```
SOMEXTERN void somtfatal(char *file, long lineno, char *fmt, ...);
SOMEXTERN void SOMLINK somtfatalSL(char *file, long lineno, char *fmt, ...);
```

Note: somtfatal version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtinternal, somtinternalSL function

```
SOMEXTERN void somtinternal(char *file, long lineno, char *fmt, ...);
SOMEXTERN void SOMLINK somtinternalSL(char *file, long lineno, char *fmt,
...);
```

Note: somtinternal version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtmclose, somtmcloseSL function

```
SOMEXTERN void somtmclose(void);
SOMEXTERN void SOMLINK somtmcloseSL(void);
```

Note: somtmclose version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtmsg, somtmsgSL function

```
SOMEXTERN void somtmsg(char *file, long lineno, char *fmt, ...);
SOMEXTERN void SOMLINK somtmsgSL(char *file, long lineno, char *fmt, ...);
```

Note: somtmsg version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtreadDescFile, somtreadDescFileSL function

```
SOMEXTERN void somtreadDescFile(Stab *stab, char *file);
SOMEXTERN void SOMLINK somtreadDescFileSL(Stab *stab, char *file);
```

Note: somtreadDescFile version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtsetDefaultDesc, somtsetDefaultDescSL function

```
SOMEXTERN void somtsetDefaultDesc(Stab *stab);
```

```
SOMEXTERN void SOMLINK somtsetDefaultDescSL(Stab *stab);
```

Note: somtsetDefaultDesc version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtsetEmitSignals, somtsetEmitSignalsSL function

```
SOMEXTERN void somtsetEmitSignals(void(*cleanup) (int), void (*internal)
(int));
SOMEXTERN void SOMLINK somtsetEmitSignalsSL(void(*cleanup) (int), void
(*internal) (int));
```

Note: somtsetEmitSignals version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtsetTypeDefn, somtsetTypeDefnSL function

```
SOMEXTERN void somtsetTypeDefn(Entry *type, Entry *ep, char *ptrs, Entry
*ret, BOOL array);
SOMEXTERN void SOMLINK somtsetTypeDefnSL(Entry *type, Entry *ep, char *ptrs,
Entry *ret, BOOL array);
```

Note: somtsetTypeDefn version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtshowVersion, somtshowVersionSL function

```
SOMEXTERN void somtshowVersion(char *s, char *progname, char *sccsid);
SOMEXTERN void SOMLINK somtshowVersionSL(char *s, char *progname, char
*sccsid);
```

Note: somtshowVersion version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtsmfree, somtsmfreeSL function

```
SOMEXTERN void somtsmfree(void *first, ...);
SOMEXTERN void SOMLINK somtsmfreeSL(void *first, ...);
```

Note: somtsmfree version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtunsetEmitSignals, somtunsetEmitSignalsSL function

```
SOMEXTERN void somtunsetEmitSignals(void);
SOMEXTERN void SOMLINK somtunsetEmitSignalsSL(void);
```

Note: somtunsetEmitSignals version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtwarn, somtwarnSL function

```
SOMEXTERN void somtwarn(char *file, long lineno, char *fmt, ...);
SOMEXTERN void SOMLINK somtwarnSL(char *file, long lineno, char *fmt, ...);
```

Note: somtwarn version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtuppercase, somtuppercaseSL function

```
SOMEXTERN char * somtuppercase(char *s, char *buf);
SOMEXTERN char * SOMLINK somtuppercaseSL(char *s, char *buf);
```

Convert <s> to upper case and return to <buf>.

Note: somtuppercase version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtlowercase, somtlowercaseSL function

```
SOMEXTERN char * somtlowercase(char *s, char *buf);
SOMEXTERN char * SOMLINK somtlowercase(char *s, char *buf)
```

Convert <s> to lower case and return to <buf>.

Note: somtlowercase version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtdbcsuppercase, somtdbcsuppercaseSL function

```
SOMEXTERN char * somtdbcsuppercase(char *s, char *buf);
SOMEXTERN char * SOMLINK somtdbcsuppercaseSL(char *s, char *buf);
```

Note: somtdbcsuppercase version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtdbcslowercase, somtdbcslowercaseSL function

```
SOMEXTERN char * somtdbcslowercase(char *s, char *buf);
```

```
SOMEXTERN char * SOMLINK somtdbcslowercaseSL(char *s, char *buf);
```

Note: somtdbcslowercase version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtresetEmitSignals, somtresetEmitSignalsSL function

```
SOMEXTERN void somtresetEmitSignals(void);
SOMEXTERN void SOMLINK somtresetEmitSignalsSL(void);
```

Note: somtresetEmitSignals version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtsizeofEntry, somtsizeofEntrySL function

```
SOMEXTERN size_t somtsizeofEntry(SOMTTypes type);
SOMEXTERN size_t SOMLINK somtsizeofEntrySL(SOMTTypes type);
```

Return size of Entry structure for <type> of entry;

Note: somtsizeofEntry version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

Mapping of type to Entry.u type and nameis following:

Entry type	**union struct**	**union name**
SOMTClassE	Class	c
SOMTMetaE	Meta	mt
SOMTBaseE	Parent	p
SOMTPassthruE	Passthru	pt
SOMTNewMethodE	Method_OR_Data	m
SOMTOverrideMethodE	Method_OR_Data	m
SOMTOverriddenMethodE	Method_OR_Data	m
SOMTDataE	Method_OR_Data	m
SOMTArgumentE	Method_OR_Data	m
SOMTTypedefBE	Method_OR_Data	m
SOMTVoidPtrBE	Method_OR_Data	m
SOMTStructE	Struct	struc
SOMTTyDclE	Typedef	ty
SOMTTypedefE	Typedef	ty
SOMTUnionE	Union	un
SOMTUnionSE	Union	un
SOMTEnumE	Enumerator	enumerator
SOMTConstE	Const	con
SOMTAttE	Att	att
SOMTSequenceE	Sequence	seq

Entry type	union struct	union name
SOMTSequenceTDE	Sequence	seq
SOMTStringE	String	str
SOMTEnumBE	EnumName	enumN
SOMTModuleE	Module	mod

somtepname, somtepnameSL function

```
SOMEXTERN char * somtepname(Entry *ep, char *buf, BOOL suppressImpctxCheck);
SOMEXTERN char * SOMLINK somtepnameSL(Entry *ep, char *buf, BOOL
suppressImpctxCheck);
```

Note: somtepname version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtgenSeqName, somtgenSeqNameSL function

```
SOMEXTERN char * somtgenSeqName(long n, Entry *base, char *buf, BOOL
fullname);
SOMEXTERN char * SOMLINK somtgenSeqNameSL(long n, Entry *base, char *buf,
BOOL fullname);
```

Note: somtgenSeqName version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtmrifatal, somtmrifatalSL function

```
SOMEXTERN void somtmrifatal(char *file, long lineno, int msgnum,...);
SOMEXTERN void SOMLINK somtmrifatalSL(char *file, long lineno, int
msgnum,...);
```

Note: somtmrifatal version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtmriinternal, somtmriinternalSL function

```
SOMEXTERN void somtmriinternal(char *file, long lineno, int msgnum,...);
SOMEXTERN void SOMLINK somtmriinternalSL(char *file, long lineno, int
msgnum,...);
```

Note: somtmriinternal version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtmrierror, somtmrierrorSL function

```
SOMEXTERN void somtmrierror(char *file, long lineno, int msgnum,...);
SOMEXTERN void SOMLINK somtmrierrorSL(char *file, long lineno, int
msgnum,...);
```

Note: somtmrierror version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtmrimsg, somtmrimsgSL function

```
SOMEXTERN void somtmrimsg(char *file, long lineno, int msgnum,...);
SOMEXTERN void SOMLINK somtmrimsgSL(char *file, long lineno, int
msgnum,...);
```

Note: somtmrimsg version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtmriwarn, somtmriwarnSL function

```
SOMEXTERN void somtmriwarn(char *file, long lineno, int msgnum,...);
SOMEXTERN void SOMLINK somtmriwarnSL(char *file, long lineno, int
msgnum,...);
```

Note: somtmriwarn version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtsetInternalMessages, somtsetInternalMessagesSL function

```
SOMEXTERN void somtsetInternalMessages(char *too_long, char *cant_continue,
char *segv, char *bus);
SOMEXTERN void SOMLINK somtsetInternalMessagesSL(char *too_long, char
*cant_continue, char *segv, char *bus);
```

Note: somtsetInternalMessages version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtisvoid, somtisvoidSL function

```
SOMEXTERN boolean somtisvoidSL(Entry *type, char *defn)
SOMEXTERN BOOL SOMLINK somtisvoidSL(Entry *type, char *defn)
```

Return TRUE if type->type is SOMTVoidBE it defn equal to “void”, “VOID”, “PMVOID”.

Note: somtisvoid version uses default compiler calling convention. For IBM SOM 3.0 for NT it is

Optlink.

somtreturnsStruct, somtreturnsStructSL function

```
SOMEXTERN BOOL somtreturnsStruct(Entry *ep);
SOMEXTERN BOOL SOMLINK somtreturnsStructSL(Entry *ep);
```

Note: somtreturnsStruct version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtreturnsPtr, somtreturnsPtrSL function

```
SOMEXTERN BOOL somtreturnsPtr(Entry *ep);
SOMEXTERN BOOL SOMLINK somtreturnsPtrSL(Entry *ep);
```

Note: somtreturnsPtr version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtsimpleName, somtsimpleNameSL function

```
SOMEXTERN char * somtsimpleName(Entry *ep);
SOMEXTERN char * SOMLINK somtsimpleNameSL(Entry *ep);
```

Note: somtsimpleName version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtqualifyNames, somtqualifyNamesSL function

```
SOMEXTERN void somtqualifyNames(Stab * stab, BOOL fully);
SOMEXTERN void SOMLINK somtqualifyNamesSL(Stab * stab, BOOL fully);
```

Note: somtqualifyNames version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtfindBaseEpNonPtr, somtfindBaseEpNonPtrSL function

```
SOMEXTERN Entry * somtfindBaseEpNonPtr(Entry *ep);
SOMEXTERN Entry * SOMLINK somtfindBaseEpNonPtrSL(Entry *ep);
```

Note: somtfindBaseEpNonPtr version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtprocessTraps, somtprocessTrapsSL function

```
SOMEXTERN BOOL somtprocessTraps(void);
SOMEXTERN BOOL SOMLINK somtprocessTrapsSL(void);
```

Note: somtprocessTraps version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtallocMlist, somtallocMlistSL function

```
SOMEXTERN Mlist * somtallocMlist(Entry * ep);
SOMEXTERN Mlist * SOMLINK somtallocMlistSL(Entry * ep);
```

Note: somtallocMlist version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtmlistend, somtmlistendSL function

```
SOMEXTERN Mlist * somtmlistend(Mlist * mp, char *name);
SOMEXTERN Mlist * SOMLINK somtmlistendSL(Mlist * mp, char *name);
```

Note: somtmlistend version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtisMutRef, somtisMutRefSL function

```
SOMEXTERN BOOL somtisMutRef(Entry *ep, Mlist *seen, BOOL isself, long
level);
SOMEXTERN BOOL SOMLINK somtisMutRefSL(Entry *ep, Mlist *seen, BOOL isself,
long level);
```

Note: somtisMutRef version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtfreeMlist, somtfreeMlistSL function

```
SOMEXTERN Mlist * somtfreeMlist(Mlist *mp);
SOMEXTERN Mlist * SOMLINK somtfreeMlistSL(Mlist *mp);
```

Note: somtfreeMlist version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtdupMlist, somtdupMlistSL function

```
SOMEXTERN Mlist * somtdupMlist(Mlist *mp, Entry *ep);
SOMEXTERN Mlist * SOMLINK somtdupMlistSL(Mlist *mp, Entry *ep);
```

Note: somtdupMlist version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtfreeWorld, somtfreeWorldSL function

```
SOMEXTERN void somtfreeWorld();
SOMEXTERN void SOMLINK somtfreeWorldSL();
```

somtinitMalloc, somtinitMallocSL function

```
SOMEXTERN void somtinitMalloc(BOOL dynamic)
SOMEXTERN void SOMLINK somtinitMallocSL(BOOL dynamic)
```

Initialize memory allocation/free functions.

Note: <dynamic> flag ignored in somFree version.

Note: somtinitMalloc version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtInitialiseEmitlib. somtInitialiseEmitlibSL function

```
SOMEXTERN void somtInitialiseEmitlib(void);
SOMEXTERN void SOMLINK somtInitialiseEmitlibSL(void);
```

Note: somtInitialiseEmitlib version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtInitialiseSmmeta, somtInitialiseSmmetaSL function

```
SOMEXTERN void somtInitialiseSmmeta(void);
SOMEXTERN void SOMLINK somtInitialiseSmmetaSL(void);
```

Note: somtInitialiseSmmeta version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtInitialiseCreatetc, somtInitialiseCreatetcSL function

```
SOMEXTERN void somtInitialiseCreatetc(void);
```

```
SOMEXTERN void SOMLINK somtInitialiseCreatetcSL(void);
```

Note: somtInitialiseCreatetc version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtInitialiseSmtypes, somtInitialiseSmtypesSL function

```
SOMEXTERN void somtInitialiseSmtypes(void);
SOMEXTERN void SOMLINK somtInitialiseSmtypesSL(void);
```

Note: somtInitialiseSmtypes version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtInitialiseSomc, somtInitialiseSomcSL function

```
SOMEXTERN void somtInitialiseSomc(void);
SOMEXTERN void SOMLINK somtInitialiseSomcSL(void);
```

Note: somtInitialiseSomc version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtInitialiseSmsmall, somtInitialiseSmsmallSL function

```
SOMEXTERN void somtInitialiseSmsmall(void);
SOMEXTERN void SOMLINK somtInitialiseSmsmallSL(void);
```

Note: somtInitialiseSmsmall version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtattMap, somtattMapSL function

somtexit, somtexitSL function

```
SOMEXTERN void somtexit(SOMTExitBuf *ebuf, int status);
SOMEXTERN void SOMLINK somtexitSL(SOMTExitBuf *ebuf, int status);
```

Note: somtexit version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtdymain, somtdymainSL function

```
SOMEXTERN void somtdymain(char *file, Entry *cls, EmitFn emitfn, char
*emitter, int first, char *version, Stab *stab);
SOMEXTERN void SOMLINK somtdymainSL(char *file, Entry *cls, EmitFn emitfn,
```

```
char *emitter, int first, char *version, Stab *stab);
```

Note: somtdymain version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtaddHeader, somtaddHeaderSL function

```
SOMEXTERN void somtaddHeader(char *file, FILE *fp, char *ext);
SOMEXTERN void SOMLINK somtaddHeaderSL(char *file, FILE *fp, char *ext);
```

Note: somtaddHeader version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtnthArg, somtnthArgSL function

```
SOMEXTERN Entry * somtnthArg(Entry * method, int n);
SOMEXTERN Entry * SOMLINK somtnthArgSL(Entry * method, int n);
```

Note: somtnthArg version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtemitModule, somtemitModuleSL function

```
SOMEXTERN FILE * somtemitModule(char *file, Entry *cls, char *ext);
SOMEXTERN FILE * SOMLINK somtemitModuleSL(char *file, Entry *cls, char
*ext);
```

Same as somtopenEmitFile.

Note: somtemitModule version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtallocDataList, somtallocDataListSL function

```
SOMEXTERN Mlist * somtallocDataList(Entry *cls);
SOMEXTERN Mlist * SOMLINK somtallocDataListSL(Entry *cls);
```

Note: somtallocDataList version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtallocMethodList, somtallocMethodListSL function

```
SOMEXTERN Mlist * somtallocMethodList(Entry *cls, boolean all);
```

```
SOMEXTERN Mlist * SOMLINK somtallocMethodListSL(Entry *cls, boolean all);
```

Note: somtallocMethodList version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtclsfilename, somtclsfilenameSL function

```
SOMEXTERN char * somtclsfilename(Entry * cls);
SOMEXTERN char * SOMLINK somtclsfilenameSL(Entry * cls);
```

Note: somtclsfilename version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtclsname, somtclsnameSL function

```
SOMEXTERN char * somtclsname(Entry * cls);
SOMEXTERN char * SOMLINK somtclsnameSL(Entry * cls);
```

Return name of class <cls>.

Note: somclsname version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtfindMethodName, somtfindMethodNameSL function

```
SOMEXTERN char * somtfindMethodName(const char *bp, char *name);
SOMEXTERN char * SOMLINK somtfindMethodNameSL(const char *bp, char *name);
```

Note: somtfindMethodName version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtfullPrototype, somtfullPrototypeSL function

```
SOMEXTERN char * somtfullPrototype(char *buf, Entry * method, char *sep, int
varargs);
SOMEXTERN char * SOMLINK somtfullPrototypeSL(char *buf, Entry * method, char
*sep, int varargs);
```

Note: somtfullPrototype version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtfullTypedef, somtfullTypedefSL function

```
SOMEXTERN char * somtfullTypedef(char *buf, Entry * cls, Entry * method);
```

```
SOMEXTERN char * SOMLINK somtfullTypedefSL(char *buf, Entry * cls, Entry *
method);
```

Note: somtfullTypedef version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtgetNonRepeatedParent, somtgetNonRepeatedParentSL function

```
SOMEXTERN char * somtgetNonRepeatedParent(Entry *cls, int i);
SOMEXTERN char * SOMLINK somtgetNonRepeatedParentSL(Entry *cls, int i);
```

Note: somtgetNonRepeatedParent version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtgetatt, somtgetattSL function

```
SOMEXTERN char * somtgetatt(Entry * ep, char *s);
SOMEXTERN char * SOMLINK somtgetattSL(Entry * ep, char *s);
```

somtgetdatt, somtgetdattSL function

```
SOMEXTERN char * somtgetdatt(Entry * ep, char *s);
SOMEXTERN char * SOMLINK somtgetdattSL(Entry * ep, char *s);
```

somtgetAbistyle, somtgetAbistyleSL function

```
SOMEXTERN enum SOMTABIStyle  somtgetAbistyle( Entry * ep );
SOMEXTERN enum SOMTABIStyle  SOMLINK somtgetAbistyleSL( Entry * ep );
```

Return ABI style of Entry. At the current time returns always SOMTABIStyle_2

Note: somtgetABIStyle version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtimplicit, somtimplicitSL function

```
SOMEXTERN char * somtimplicit(Entry *ep, boolean shortform, char *buf);
SOMEXTERN char * SOMLINK somtimplicitSL(Entry *ep, boolean shortform, char
*buf);
```

Note: somtimplicit version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtimplicitArgs, somtimplicitArgsSL function

```
SOMEXTERN char * somtimplicitArgs(Entry *ep);
SOMEXTERN char * SOMLINK somtimplicitArgsSL(Entry *ep);
```

Note: somtimplicitArgs version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtincludeOnce, somtincludeOnceSL function

```
SOMEXTERN char * somtincludeOnceSL(Entry *cls, char *ext, char *buf);
SOMEXTERN char * SOMLINK somtincludeOnceSL(Entry *cls, char *ext, char
*buf);
```

Return token to <buf> for once include checks using name of class <cls> and extension <ext> in form SOM_classname_ext.

Note: somtincludeOnce version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtpclsfilename, somtpclsfilenameSL function

```
SOMEXTERN char * somtpclsfilename(Entry *parent);
SOMEXTERN char * SOMLINK somtpclsfilenameSL(Entry *parent);
```

Note: somtpclsfilename version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtpclsname, somtpclsnameSL function

```
SOMEXTERN char * somtpclsname(Entry *parent);
SOMEXTERN char * SOMLINK somtpclsnameSL(Entry *parent);
```

Note: somtpclsname version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtprefixedPrototype, somtprefixedPrototypeSL function

```
SOMEXTERN char * somtprefixedPrototype(char *buf, Entry * method, char *sep,
int varargs, char *prefix);
SOMEXTERN char * SOMLINK somtprefixedPrototypeSL(char *buf, Entry * method,
char *sep, int varargs, char *prefix);
```

Note: somtprefixedPrototype version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtreplaceDataName, somtreplaceDataNameSL function

```
SOMEXTERN char * somtreplaceDataName(char *buf, Entry * data, char
*replace);
SOMEXTERN char * SOMLINK somtreplaceDataNameSL(char *buf, Entry * data, char
*replace);
```

Note: somtreplaceDataName version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtrmSelf, somtrmSelfSL function

```
SOMEXTERN char * somtrmSelf(char *str);
SOMEXTERN char * SOMLINK somtrmSelfSL(char *str);
```

Note: somtrmSelf version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtshortArgList, somtshortArgListSL function

```
SOMEXTERN char * somtshortArgList(char *buf, Entry * method, char *sep,
boolean varargs, boolean addself);
SOMEXTERN char * SOMLINK somtshortArgListSL(char *buf, Entry * method, char
*sep, boolean varargs, boolean addself);
```

Note: somtshortArgList version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtimplicitMeta, somtimplicitMetaSL function

```
SOMEXTERN int  somtimplicitMeta(Entry *cls);
SOMEXTERN int  SOMLINK somtimplicitMetaSL(Entry *cls);
```

Note: somtimplicitMeta version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtlistAttribute, somtlistAttributeSL function

```
SOMEXTERN int  somtlistAttribute(FILE * fp, int n, AttList * ap, char *s,
boolean value, boolean breakLine, boolean firstComma);
SOMEXTERN int  SOMLINK somtlistAttributeSL(FILE * fp, int n, AttList * ap,
char *s, boolean value, boolean breakLine, boolean firstComma);
```

Note: somtlistAttribute version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtnewMethodsCount, somtnewMethodsCountSL function

```
SOMEXTERN int  somtnewMethodsCount(Entry * cls, int meta, boolean procflg);
SOMEXTERN int  SOMLINK somtnewMethodsCountSL(Entry * cls, int meta, boolean
procflg);
```

Note: somtnewMethodsCount version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtprivateMethodsCount, somtprivateMethodsCountSL function

```
SOMEXTERN int somtprivateMethodsCount(Entry * cls, int meta);
SOMEXTERN int SOMLINK somtprivateMethodsCountSL(Entry * cls, int meta);
```

Note: somtprivateMethodsCount version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtaddHeader, somtaddHeaderSL function

```
SOMEXTERN void somtaddHeader(char *file, FILE *fp, char *ext);
SOMEXTERN void SOMLINK somtaddHeaderSL(char *file, FILE *fp, char *ext);
```

Note: somtaddHeader version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtcleanFiles, somtcleanFilesSL function

```
SOMEXTERN void somtcleanFiles(int status);
SOMEXTERN void  SOMLINK somtcleanFilesSL(int status);
```

Delete temporary files and exit.

Note: somtcleanFiles version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtdeclareIdlVarargs, somtdeclareIdlVarargsSL function

```
SOMEXTERN void somtdeclareIdlVarargs(FILE *fp, Entry *ep);
SOMEXTERN void SOMLINK somtdeclareIdlVarargsSL(FILE *fp, Entry *ep);
```

Note: somtdeclareIdlVarargs version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtdymain. somtdymainSL function

```
SOMEXTERN void somtdymain(char *file, Entry *cls, EmitFn emitfn, char
*emitter, int first, char *version, Stab *stab);
SOMEXTERN void SOMLINK somtdymainSL(char *file, Entry *cls, EmitFn emitfn,
char *emitter, int first, char *version, Stab *stab);
```

Note: somtdymain version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtemitModuleTypes, somtemitModuleTypesSL function

```
SOMEXTERN void somtemitModuleTypes(FILE *fp, Entry *ep, Stab *stab);
SOMEXTERN void SOMLINK somtemitModuleTypesSL(FILE *fp, Entry *ep, Stab
*stab);
```

Note: somtemitModuleTypes version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtemitPassthru, somtemitPassthruSL function

```
SOMEXTERN long somtemitPassthru(FILE * fp, Entry * cls, char *name, int
mode, char *att);
SOMEXTERN long SOMLINK somtemitPassthruSL(FILE * fp, Entry * cls, char
*name, int mode, char *att);
```

Note: somtemitPassthru version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtfreeDataList, somtfreeDataListSL function

```
SOMEXTERN void somtfreeDataList(Mlist *mlist);
SOMEXTERN void SOMLINK somtfreeDataListSL(Mlist *mlist);
```

Note: somtfreeDataList version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtfreeMethodList, somtfreeMethodListSL function

```
SOMEXTERN void somtfreeMethodList(Mlist *mlist);
SOMEXTERN void SOMLINK somtfreeMethodListSL(Mlist *mlist);
```

Note: somtfreeMethodList version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtfullComment, somtfullCommentSL function

```
SOMEXTERN void somtfullCommentSL(FILE * fp, char *fmt,...);
SOMEXTERN void SOMLINK somtfullCommentSL(FILE * fp, char *fmt,...);
```

Outout formatted string <fmt> to emitted file as comment using C-style comment via somtoidlComment function;

Note: somtfullComment version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somthandleDiskFull, somthandleDiskFullSL function

```
SOMEXTERN void somthandleDiskFull(FILE *fp);
SOMEXTERN void SOMLINK somthandleDiskFullSL(FILE *fp);
```

Note: somthandleDiskFull version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtinitialiseMeta, somtinitialiseMetaSL function

```
SOMEXTERN void somtinitialiseMeta(Entry * cls, Stab * stab, boolean meta,
int imp);
SOMEXTERN void SOMLINK somtinitialiseMetaSL(Entry * cls, Stab * stab,
boolean meta, int imp);
```

Note: somtinitialiseMeta version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtoidlComment, somtoidlCommentSL function

```
SOMEXTERN void somtoidlComment(FILE * fp, int min, int max, char style, char
*comment);
SOMEXTERN void  SOMLINK somtoidlCommentSL(FILE * fp, int min, int max, char
style, char *comment);
```

Output oidl-<style> <comment> to file <fp> from colon <min> up to colon <max>.

Note: Seems IBM SOM ignores <max> value.

Style is one of following:

- '/' - each line started from "//#";
- '#' - each line started from “#”;
- 'c' - C-style comment started from '/*' and ended with '*/'. each line started from "*";
- 's' -each line started from "--";
- 'd' - each line started from “;”;

- '+' - each line started from "//";

Other values forced to 'c' style.

<comment> can contains at offset 0 0x01 signature indicating comment style. If style is zero the used style from comment position 1. Two first symbols of comment are ignored if style signature is present.

Note: somtoidlComment version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtscmsg, somtscmsgSL function

```
SOMEXTERN void somtscmsg(Entry *cls, Entry *ep, char *fmt, ...);
SOMEXTERN void SOMLINK somtscmsgSL(Entry *cls, Entry *ep, char *fmt, ...);
```

Note: somtscmsg version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtshortDefine, somtshortDefineSL function

```
SOMEXTERN void somtshortDefine(FILE *fp, Entry *ep, char *fmt, ...);
SOMEXTERN void SOMLINK somtshortDefineSL(FILE *fp, Entry *ep, char *fmt,
...);
```

Note: somtshortDefine version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtuninitialiseMeta, somtuninitialiseMetaSL function

```
SOMEXTERN void  somtuninitialiseMeta(Entry * cls);
SOMEXTERN void SOMLINK somtuninitialiseMetaSL(Entry * cls);
```

Note: somtuninitialiseMeta version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtobseleteHeaderFile, somtobseleteHeaderFileSL function

```
SOMEXTERN FILE * somtobseleteHeaderFile(char *file, Entry *cls, char *ext,
char *newext);
SOMEXTERN FILE * SOMLINK somtobseleteHeaderFileSL(char *file, Entry *cls,
char *ext, char *newext);
```

Open emit file and write info about obsolete header. Return file pointer.

Note: somtoboleteHeaderFile version uses default compiler calling convention. For IBM SOM 3.0 for NT

it is Optlink.

somtwidenType, somtwidenTypeSL function

```
SOMEXTERN char * somtwidenType(Entry *ep, char *args, char *type);
SOMEXTERN char * SOMLINK somtwidenTypeSL(Entry *ep, char *args, char *type);
```

Note: somtwidenType version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtgenAttStubs, somtgenAttStubsSL function

```
SOMEXTERN void somtgenAttStubs(FILE *fp, Entry *cls, char *prefix, char
*classprefix);
SOMEXTERN void SOMLINK somtgenAttStubsSL(FILE *fp, Entry *cls, char *prefix,
char *classprefix);
```

Note: somtgenAttStubs version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtstrictidl, somtstrictidlSL function

```
SOMEXTERN void somtstrictidl(FILE *fp);
SOMEXTERN void SOMLINK somtstrictidlSL(FILE *fp);
```

Output definition of SOM_STRICT_IDL macro if somadd variable is TRUE;

Note: somtstrictidl version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtcreateTypeCodes, somtcreateTypeCodesSL function

```
SOMEXTERN void somtcreateTypeCodes (Stab *stab);
SOMEXTERN void SOMLINK somtcreateTypeCodesSL(Stab *stab);
```

Note: somtcreateTypeCodes version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtemitTcConstant, somtemitTcConstantSL function

```
SOMEXTERN TypeCode * somtemitTcConstant(TypeCode t, FILE *f, char *name,
TypeCode *alreadyDone);
SOMEXTERN TypeCode * SOMLINK somtemitTcConstantSL(TypeCode t, FILE *f, char
*name, TypeCode *alreadyDone);
```

Note: somtemitTcConstant version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtemitPredefinedTcConstants, somtemitPredefinedTcConstantsSL function

```
SOMEXTERN void somtemitPredefinedTcConstants (FILE *f);
SOMEXTERN void SOMLINK somtemitPredefinedTcConstantsSL(FILE *f);
```

Note: somtemitPredefinedTcConstants version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtAncestorClass, somtAncestorClassSL function

```
SOMEXTERN Entry * somtAncestorClass(Entry *cls, char *name);
SOMEXTERN Entry * SOMLINK somtAncestorClassSL(Entry *cls, char *name);
```

Note: somtAncestorClass version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somttcAlignment, somttcAlignmentSL function

```
SOMEXTERN short somttcAlignment (TypeCode t, Environment *ev);
SOMEXTERN short SOMLINK somttcAlignmentSL(TypeCode t, Environment *ev);
```

Note: somttcAlignment version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somttcSize, somttcSizeSL function

```
SOMEXTERN long somttcSize (TypeCode t, Environment *ev);
SOMEXTERN long SOMLINK somttcSizeSL(TypeCode t, Environment *ev);
```

Note: somttcSize version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somttcKind, somttcKindSL function

```
SOMEXTERN TCKind somttcKind (TypeCode t, Environment *ev);
SOMEXTERN TCKind SOMLINK somttcKindSL(TypeCode t, Environment *ev);
```

Note: somttcKind version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somttcSeqFromListString, somttcSeqFromListStringSL function

```
SOMEXTERN sequence(string) somttcSeqFromListString (string s);
SOMEXTERN sequence(string) SOMLINK somttcSeqFromListStringSL(string s);
```

Note: somttcSeqFromListString version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtGetReintroducedMethods, somtGetReintroducedMethodsSL function

```
SOMEXTERN _IDL_SEQUENCE_EntryPtr somtGetReintroducedMethods(Entry *cls);
SOMEXTERN _IDL_SEQUENCE_EntryPtr SOMLINK somtGetReintroducedMethodsSL(Entry
*cls);
```

Note: somtGetReintroducedMethods version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

Symbol table support functions

somtallocBuf, somtallocBufSL function

Note: somtallocBuf version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtuniqString, somtuniqStringSL function

```
SOMEXTERN char * somtuniqString(MemBuf *membuf, char *s);
SOMEXTERN char * SOMLINK somtuniqStringSL(MemBuf *membuf, char *s);
```

Check is string unique and return NULL if not, or string itself if unique;

Note: somtuniqString version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtkeyword, somtkeywordSL function

```
SOMEXTERN long somtkeyword(KeytabEntry *keytab, char *kword, long
keytabsize);
SOMEXTERN long SOMLINK somtkeywordSL(KeytabEntry *keytab, char *kword, long
keytabsize);
```

Return token for keyword <kword> from keytaable <keytab> of <keytabsize> size.

Note: somtkeyword version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtaddEntry, somtaddEntrySL function

```
SOMEXTERN void * somtaddEntry(Stab *stab, char *name, void *ep);
SOMEXTERN void * SOMLINK somtaddEntrySL(Stab *stab, char *name, void *ep);
```

Add entry <ep> with name <name> to symbol table <stab>. Buffer for entry allocated by function.

Note: somtaddEntry version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtgetEntry, somtgetEntrySL function

```
SOMEXTERN void * somtgetEntry(Stab *stab, char *name);
SOMEXTERN void * SOMLINK somtgetEntrySL(Stab *stab, char *name);
```

Return pointer to entry structure with name equal to <name> from symbol table <stab>

Note: somtgetEntry version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtstabFirst, somtstabFirstSL function

```
SOMEXTERN void * somtstabFirst(Stab *stab, Sep **sepp);
SOMEXTERN void * SOMLINK somtstabFirstSL(Stab *stab, Sep **sepp);
```

Return first entry from symbol table <stab> and, optionally, returns sep entry in <sepp>.

Note: somtstabFirst version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtstabNext, somtstabNextSL function

```
SOMEXTERN void * somtstabNext(Stab *stab, Sep **sepp);
SOMEXTERN void * SOMLINK somtstabNextSL(Stab *stab, Sep **sepp);
```

Return next after last search entry from symbol table <stab> and, optionally, returns sep entry in <sepp>.

Note: somtstabNext version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtstabFirstName, somtstabFirstNameSL function

```
SOMEXTERN void * somtstabFirstName(Stab *stab, char *name, Sep **sepp);
SOMEXTERN void * SOMLINK somtstabFirstNameSL(Stab *stab, char *name, Sep
**sepp);
```

Return first entry with <name> from symbol table <stab> and, optionally, returns sep entry in <sepp>.

Note: somtstabFirstName version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtstabNextName, somtstabNextNameSL function

```
SOMEXTERN void * somtstabNextName(Stab *stab, Sep **sepp);
SOMEXTERN void * SOMLINK somtstabNextNameSL(Stab *stab, Sep **sepp);
```

Return next after last search entry from symbol table <stab> and, optionally, returns sep entry in <sepp>.

Note: somtstabNextName version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtcreateMemBuf, somtcreateMemBufSL function

```
SOMEXTERN void somtcreateMemBuf(MemBuf **membufp, size_t bufsize, long
stabsize);
SOMEXTERN void SOMLINK somtcreateMemBufSL(MemBuf **membufp, size_t bufsize,
long stabsize);
```

Note: somtcreateMemBuf version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtcreateStab, somtcreateStabSL function

```
SOMEXTERN void somtcreateStab(Stab *stab, long stabsize, long entrysize);
SOMEXTERN void SOMLINK somtcreateStabSL(Stab *stab, long stabsize, long
entrysize);
```

Initialize symbol table structure <stab> using hash index size <stabsize> and entry size <entrysize>.

Note: somtcreateStab version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somticstrcmp, somticstrcmpSL function

```
SOMEXTERN int somticstrcmp(char *s, char *t)
SOMEXTERN int SOMLINK somticstrcmpSL(char *s, char *t);
```

Alias of C stricmp.

Note: somticstrcmp version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtaddEntryBuf, somtaddEntryBufSL function

```
SOMEXTERN void * somtaddEntryBuf(Stab *stab, char *name, void *ep, void
*buf, size_t len);
SOMEXTERN void * SOMLINK somtaddEntryBufSL(Stab *stab, char *name, void *ep,
void *buf, size_t len);
```

Add entry <ep> with name <name> to symbol table <stab> to buffer <buf> with size <len>

Note: somtaddEntryBuf version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

somtfreeStab, somtfreeStabSL function

```
SOMEXTERN void somtfreeStab(Stab *stab, BOOL freeEp);
SOMEXTERN void SOMLINK somtfreeStabSL(Stab *stab, BOOL freeEp);
```

Note: somtfreeStab version uses default compiler calling convention. For IBM SOM 3.0 for NT it is Optlink.

3. somFree Emitter Framework

SOMTAttributeEntryC Class

somtIsReadonly attribute

```
readonly attribute boolean somtIsReadonly;
```

Whether the attribute is readonly.

somtAttribType attribute

```
readonly attribute SOMTEntryC somtAttribType;
```

The type of the attribute. This does not include pointer stars or array declarators. To get the “full” type, get each attribute declarator and get the somtType attribute.

somtGetFirstAttributeDeclarator method

```
SOMTDataEntryC somtGetFirstAttributeDeclarator();
```

The first attribute declarator for this attribute declaration.

somtGetNextAttributeDeclarator method

```
SOMTDataEntryC somtGetNextAttributeDeclarator();
```

The next attribute declarator for this attribute declaration, relative to the previous call to this method or somtGetFirstAttributeDeclarator.

somtGetFirstGetMethod method

```
SOMTMethodEntryC somtGetFirstGetMethod();
```

The first get method for this attribute declaration.

somtGetNextGetMethod method

```
SOMTMethodEntryC somtGetNextGetMethod();
```

The next get method for this attribute declaration, relative to the previous call to this method or somtGetFirstGetMethod.

somtGetFirstSetMethod method

```
SOMTMethodEntryC somtGetFirstSetMethod();
```

The first set method for this attribute declaration.

somtGetNextSetMethod method

```
SOMTMethodEntryC somtGetNextSetMethod();
```

The next set method for this attribute declaration, relative to the previous call to this method or somtGetFirstSetMethod.

SOMTBaseClassEntryC Class

somtBaseClassDef attribute

```
readonly attribute SOMTClassEntryC somtBaseClassDef;
```

Returns the class definition entry for the Base class named in this entry.

SOMTClassEntryC Class

somtSourceFileName attribute

```
readonly attribute string somtSourceFileName;
```

Returns the name of file containing the definition of this class.

somtMetaClassEntry attribute

```
readonly attribute SOMTMetaClassEntryC somtMetaClassEntry;
```

Returns the entry for the meta class statement in class definition or NULL if there is no meta class statement.

Note: the SOM architecture requires that all classes have a meta class, however <SOMClass> is its own metaclass. Thus, any attempt to walk up the metaclass chain must terminate when it finds a class that is its own meta class, otherwise an infinite loop is possible.

somtClassModule attribute

```
readonly attribute SOMTModuleEntryC somtClassModule;
```

The module that contains this class, or NULL if there is not one.

somtNewMethodCount attribute

```
readonly attribute long somtNewMethodCount;
```

Returns the number of new methods introduced in this class definition.

somtLocalInclude attribute

```
readonly attribute boolean somtLocalInclude;
```

Returns true if the header files associated with this class definition should be included using local search, eg, “name.h” instead of <name.h>

somtPrivateMethodCount attribute

```
readonly attribute long somtPrivateMethodCount;
```

Returns number of new private methods in class.

somtStaticMethodCount attribute

```
readonly attribute long somtStaticMethodCount;
```

Returns number of new static methods in class.

somtOverrideMethodCount attribute

```
readonly attribute long somtOverrideMethodCount;
```

Returns number of new override methods in class.

somtProcMethodCount attribute

```
readonly attribute long somtProcMethodCount;
```

Returns number of procedure methods for class.

somtVAMethodCount attribute

```
readonly attribute long somtVAMethodCount;
```

Returns number of VarArg methods for class.

somtBaseCount attribute

```
readonly attribute long somtBaseCount;
```

Returns number of base classes for class.

somtExternalDataCount attribute

```
readonly attribute long somtExternalDataCount;
```

Returns number of external (public or private) data members for class.

somtPublicDataCount attribute

```
readonly attribute long somtPublicDataCount;
```

Returns number of public data members for class.

somtPrivateDataCount attribute

```
readonly attribute long somtPrivateDataCount;
```

Returns number of private data members for class.

somtMetaclassFor attribute

```
readonly attribute SOMTClassEntryC somtMetaclassFor;
```

If this is a metaclass, the class for which it is a metaclass, else NULL.

somtForwardRef attribute

```
readonly attribute boolean somtForwardRef;
```

Whether this is a forward reference or not.

somtGetFirstBaseClass method

```
SOMTBaseClassEntryC somtGetFirstBaseClass();
```

Returns the entry for the “left most” direct base class form for this class, if it has one and NULL otherwise.

Note: <SOMObject> does not have any base classes and therefore will terminate an attempt to walk

up the base class chain.

somtGetNextBaseClass method

```
SOMTBaseClassEntryC somtGetNextBaseClass();
```

Returns the entry for the next direct base class form of this class, if it has one and NULL otherwise. The direct base classes of a derived class are ordered from “left to right”.

somtGetFirstReleaseName method

```
string somtGetFirstReleaseName();
```

Returns the first name in the release order statement for this entry if it has one and NULL otherwise.

somtGetNextReleaseName method

```
string somtGetNextReleaseName();
```

Returns the next name in the release order statement for this entry if it has one and NULL otherwise.

somtGetReleaseNameList method

```
long somtGetReleaseNameList(in string buffer);
```

Puts all the release names in <buffer> in template output form, buffer must be large enough, no tests are made. The number of release names is returned.

somtGetFirstPassthru method

```
SOMTPassthruEntryC somtGetFirstPassthru();
```

Returns the first passthru entry for this class definition if it has one and NULL otherwise.

somtGetNextPassthru method

```
SOMTPassthruEntryC somtGetNextPassthru();
```

Returns the next passthru entry for this class definition if it has one and NULL otherwise. The passthru entry will be returned in an order based on the appearence of passthru statements in the class definition.

somtGetFirstData method

```
SOMTDataEntryC somtGetFirstData();
```

Returns the first data entry for this class definition if it has one and NULL otherwise.

somtGetNextData method

```
SOMTDataEntryC somtGetNextData();
```

Returns the next data entry for this class definition if it has one and NULL otherwise. The data entries will be returned in an order based on the appearence data member declarations in the class definition.

somtGetFirstStaticData method

```
SOMTDataEntryC somtGetFirstStaticData();
```

Returns the first static data entry for this class definition if it has one and NULL otherwise. Static data is handled specialy in SOM so a different accessor method is provided.

somtGetNextStaticData method

```
SOMTDataEntryC somtGetNextStaticData();
```

Returns the next static data entry for this class definition if it has one and NULL otherwise. The data entries will be returned in an order based on the release order

somtGetFirstMethod method

```
SOMTMethodEntryC somtGetFirstMethod();
```

Returns the first method entry for this class definition if it has one and NULL otherwise. Method entries may be for new or overridden methods.

somtGetNextMethod method

```
SOMTMethodEntryC somtGetNextMethod();
```

Returns the next method entry for this class definition if it has one and NULL otherwise. The method entries will be returned in an order based on the appearence method declarations in the class definition. Method entries may be for new or overridden methods.

somtGetFirstInheritedMethod method

```
SOMTMethodEntryC somtGetFirstInheritedMethod();
```

Returns the first inherited and not overridden method entry for this class definition if it has one and NULL otherwise.

somtGetNextInheritedMethod method

```
SOMTMethodEntryC somtGetNextInheritedMethod();
```

Returns the next inherited and not overridden method entry for this class definition if it has one and NULL otherwise. The method entries will be returned in an unspecified, but constant order.

somtGetFirstAttribute method

```
SOMTAttributeEntryC somtGetFirstAttribute();
```

somtGetNextAttribute method

```
SOMTAttributeEntryC somtGetNextAttribute();
```

somtGetFirstStruct method

```
SOMTStructEntryC somtGetFirstStruct();
```

somtGetNextStruct method

```
SOMTStructEntryC somtGetNextStruct();
```

somtGetFirstTypedef method

```
SOMTTypedefEntryC somtGetFirstTypedef();
```

somtGetNextTypedef method

```
SOMTTypedefEntryC somtGetNextTypedef();
```

somtGetFirstUnion method

```
SOMTUnionEntryC somtGetFirstUnion();
```

somtGetNextUnion method

```
SOMTUnionEntryC somtGetNextUnion();
```

somtGetFirstEnum method

```
SOMTEnumEntryC somtGetFirstEnum();
```

somtGetNextEnum method

```
SOMTEnumEntryC somtGetNextEnum();
```

somtGetFirstConstant method

```
SOMTConstEntryC somtGetFirstConstant();
```

somtGetNextConstant method

```
SOMTConstEntryC somtGetNextConstant();
```

somtGetFirstSequence method

```
SOMTSequenceEntryC somtGetFirstSequence();
```

somtGetNextSequence method

```
SOMTSequenceEntryC somtGetNextSequence();
```

somtGetFirstPubdef method

```
SOMTEntryC somtGetFirstPubdef();
```

somtGetNextPubdef method

```
SOMTEntryC somtGetNextPubdef();
```

somtFilterNew method

```
boolean somtFilterNew(in SOMTMethodEntryC entry);
```

Returns 1 if entry is new in the class.

somtFilterOverridden method

```
boolean somtFilterOverridden(in SOMTMethodEntryC entry);
```

Returns 1 if entry is an overriding method of the class.

somtFilterPrivOrPub method

```
boolean somtFilterPrivOrPub(in SOMTCommonEntryC entry);
```

Returns TRUE if entry is Private or Public.

SOMTCommonEntryC Class

somtTypeObj attribute

```
readonly attribute SOMTEntryC somtTypeObj;
```

The object representing the base type of the entry. This does not include pointer stars or array declarators.

somtPtrs attribute

```
readonly attribute string somtPtrs;
```

The string of stars associated with the entry's type. For example, an object of type “foo” would have somtPtrs = NULL, type “foo *” would have somtPtrs = “*”, type “foo **” would have somtPtrs = “**”, etc.

somtArrayDimsString attribute

```
readonly attribute string somtArrayDimsString;
```

Array dimensions in string form.

somtGetFirstArrayDimension method

```
unsigned long somtGetFirstArrayDimension();
```

The first array dimension, for items of type array. Zero indicates that the item is not an array.

somtGetNextArrayDimension method

```
unsigned long somtGetNextArrayDimension();
```

The next array dimension, for items of type array, relative to the previous call to this method or to somtGetFirstArrayDimension. Zero indicates no more dimensions.

somtSourceText attribute

```
readonly attribute string somtSourceText;
```

The un-parsed source text for this entry, with leading and trailing white space removed. For attribute/typedef declarators and for user-defined types, this attribute only provides the source text for the entry's name. For methods, arguments, and instance variables, however, this attribute provides the full definition.

somtType attribute

```
readonly attribute string somtType;
```

The IDL type for this entry in string form. For methods this is the return type. For data or parameters this is the type of the data item or parameter. For user-defined types, this is the type specification. It is of the form: <typename><pointer-stars> <array-declarators>

somtVisibility attribute

```
readonly attribute somtVisibilityT somtVisibility;
```

The visibility of this entry. Note: the visibility of parameter entries will always be public, and methods can never be internal.

somtIsArray method

```
boolean somtIsArray(out long size);
```

Returns 1 (true) if the type involves an array. When the type involves an array then <size> is set to be the size of the array.

somtIsPointer method

```
boolean somtIsPointer();
```

Returns 1 (true) if the type involves a pointer, and 0 (false) otherwise

SOMTConstEntryC Class

somtConstTypeObj attribute

```
readonly attribute SOMTEntryC somtConstTypeObj;
```

A pointer to an object representing the type of the const.

somtConstType attribute

```
readonly attribute string somtConstType;
```

The type of the constant's value.

somtConstStringVal attribute

```
readonly attribute string somtConstStringVal;
```

The string value of the constant (unevaluated).

somtConstNumVal attribute

```
readonly attribute unsigned long somtConstNumVal;
```

The number value of the constant. This attribute is not valid if the value cannot be stored in an unsigned long (string, float, double, negative). The somtConstIsNegative attribute can be used to determine if the value is negative. The somtConstType attribute can be used to determine whether the value is a float or double.

somtConstNumNegVal attribute

```
readonly attribute long somtConstNumNegVal;
```

The number value of the constant, if negative.

somtConstIsNegative attribute

```
readonly attribute boolean somtConstIsNegative;
```

Whether the constant's value is a negative integer and must be obtained using somtConstNumNegVal rather than somtConstNumVal.

somtConstVal attribute

```
readonly attribute string somtConstVal;
```

The string value of the constant (evaluated). The “get” method for this attribute returns a string whose ownership is transferred to the caller.

SOMTDataEntryC Class

somtIsSelfRef attribute

```
readonly attribute boolean somtIsSelfRef;
```

Whether a declarator of a struct is self-referential.

SOMTEmitC Class

somtTemplate attribute

```
attribute SOMTTemplateOutputC somtTemplate;
```

The template is to provide template output and maintains a symbol table that provides a sort of global context for the emitter.

somtTargetFile attribute

```
attribute FILE *somtTargetFile;
```

The target file is the one to which all emitter output is to be directed.

somtTargetClass attribute

```
attribute SOMTClassEntryC somtTargetClass;
```

The target class is the class definition for which code is to be emitted.

somtTargetModule attribute

```
attribute SOMTModuleEntryC somtTargetModule;
```

The target module is the module definition for which code is to be emitted.

somtTargetType attribute

```
attribute SOMTTargetTypeT somtTargetType;
```

The target type indicates what type of output file is being produced, public, private, or implementation. This allows the same emitter subclass to produce several different output files that generally differ only in how much of the class definition they cover. Eg, .csc, .sc, and .psc. This is attribute is for OIDL compatibility only.

somtEmitterName attribute

```
attribute string somtEmitterName;
```

The short name of the emitter (the name used to invoke it via the SOM Compiler. Typically this is the file stem of the subclass of SOMTEmitC. This attribute should be set in the driver program that runs the emitter. It is used to filter passthrus so that only passthrus directed to a particular emitter are seen by it.

somtGenerateSections method

```
boolean somtGenerateSections();
```

Calls each of the section methods in order. The order is:

```
   somtEmitProlog
   when emitting a class:
      somtEmitClass
      somtEmitBase
      somtEmitMeta
   somtEmitConstant
   somtEmitTypedef
   somtEmitStruct
   somtEmitUnion
   somtEmitEnum
   when emitting a class:
```

```
      somtEmitAttribute
      somtEmitMethod
      somtEmitRelease
      somtEmitPassthru
      somtEmitData
   when emitting a module:
       somtEmitInterface
       somtEmitModule
   somtEmitEpilog
```

This method will need to be overridden by many emitters in order to rearange the order of the sections and to add or delete sections.

Note: repeating sections such as methods, data, and passthru, have a prolog and epilog method as well. The prolog method is called before the first sections is processed and the epilog method is called after the last section is processed.

somtOpenSymbolsFile method

```
FILE* somtOpenSymbolsFile(in string file, in string mode);
```

This method attempts to open the symbols file. If file doesn't exist then it will attempt to find it in the directories specified in the SMINCLUDE environment variable. If the file can be found a FILE * pointer is returned, otherwise NULL is returned.

somtSetPredefinedSymbols method

```
void somtSetPredefinedSymbols();
```

Set predefined symbols that are used for such things as section names etc.

somtFileSymbols method

```
void somtFileSymbols();
```

Symbols that are common to the file. This includes the target class symbols, and the metaclass symbols, and special symbols like <timeStamp>. IE, all symbols that have a single definition.

somtEmitProlog method

```
void somtEmitProlog();
```

somtEmitBaseIncludesProlog method

```
void somtEmitBaseIncludesProlog();
```

somtEmitBaseIncludes method

```
void somtEmitBaseIncludes(in SOMTBaseClassEntryC base);
```

somtEmitBaseIncludesEpilog method

```
void somtEmitBaseIncludesEpilog();
```

somtEmitMetaInclude method

```
void somtEmitMetaInclude();
```

somtEmitClass method

```
void somtEmitClass();
```

somtEmitMeta method

```
void somtEmitMeta();
```

somtEmitBaseProlog method

```
void somtEmitBaseProlog();
```

somtEmitBase method

```
void somtEmitBase(in SOMTBaseClassEntryC base);
```

somtEmitBaseEpilog method

```
void somtEmitBaseEpilog();
```

somtEmitPassthruProlog method

```
void somtEmitPassthruProlog();
```

somtEmitPassthru method

```
void somtEmitPassthru(in SOMTPassthruEntryC entry);
```

somtEmitPassthruEpilog method

```
void somtEmitPassthruEpilog();
```

somtEmitRelease method

```
void somtEmitRelease();
```

somtEmitDataProlog method

```
void somtEmitDataProlog();
```

somtEmitData method

```
void somtEmitData(in SOMTDataEntryC entry);
```

somtEmitDataEpilog method

```
void somtEmitDataEpilog();
```

somtEmitAttributeProlog method

```
void somtEmitAttributeProlog();
```

somtEmitAttribute method

```
void somtEmitAttribute(in SOMTAttributeEntryC att);
```

somtEmitAttributeEpilog method

```
void somtEmitAttributeEpilog();
```

somtEmitConstantProlog method

```
void somtEmitConstantProlog();
```

somtEmitConstant method

```
void somtEmitConstant(in SOMTConstEntryC con);
```

somtEmitConstantEpilog method

```
void somtEmitConstantEpilog();
```

somtEmitTypedefProlog method

```
void somtEmitTypedefProlog();
```

somtEmitTypedef method

```
void somtEmitTypedef(in SOMTTypedefEntryC td);
```

somtEmitTypedefEpilog method

```
void somtEmitTypedefEpilog();
```

somtEmitStructProlog method

```
void somtEmitStructProlog();
```

somtEmitStruct method

```
void somtEmitStruct(in SOMTStructEntryC struc);
```

somtEmitStructEpilog method

```
void somtEmitStructEpilog();
```

somtEmitUnionProlog method

```
void somtEmitUnionProlog();
```

somtEmitUnion method

```
void somtEmitUnion(in SOMTUnionEntryC un);
```

somtEmitUnionEpilog method

```
void somtEmitUnionEpilog();
```

somtEmitEnumProlog method

```
void somtEmitEnumProlog();
```

somtEmitEnum method

```
void somtEmitEnum(in SOMTEnumEntryC en);
```

somtEmitEnumEpilog method

```
void somtEmitEnumEpilog();
```

somtEmitInterfaceProlog method

```
void somtEmitInterfaceProlog();
```

somtEmitInterface method

```
void somtEmitInterface(in SOMTClassEntryC intfc);
```

somtEmitInterfaceEpilog method

```
void somtEmitInterfaceEpilog();
```

somtEmitModuleProlog method

```
void somtEmitModuleProlog();
```

somtEmitModule method

```
void somtEmitModule(in SOMTModuleEntryC mod);
```

somtEmitModuleEpilog method

```
void somtEmitModuleEpilog();
```

somtEmitMethodsProlog method

```
void somtEmitMethodsProlog();
```

somtEmitMethods method

```
void somtEmitMethods(in SOMTMethodEntryC method);
```

somtEmitMethodsEpilog method

```
void somtEmitMethodsEpilog();
```

somtEmitMethod method

```
void somtEmitMethod(in SOMTMethodEntryC entry);
```

somtEmitEpilog method

```
void somtEmitEpilog();
```

somtScanBases method

```
boolean somtScanBases(in string prolog, in string each, in string epilog);
```

somtScanBaseIncludes method

```
boolean somtScanBaseIncludes(in string prolog, in string each, in string
epilog);
```

somtCheckVisibility method

```
boolean somtCheckVisibility(in SOMTMethodEntryC entry);
```

Return 1 (true) if <entry> should be visible in the current target file. This method is used by each of the following filter methods that are concerned with visibility.

The default rule for visibility is:

- only private methods are visible in private target files,
- only public methods are visibile in public target files,
- all methods are visibile in implementation or <somtAllE> target files.

somtNew method

```
boolean somtNew(in SOMTMethodEntryC entry);
```

Returns 1 (true) if <entry> is a method introduced by the target class and its visibility matches <somtTargetType> (somtImplementationE matches both private and public)

somtImplemented method

```
boolean somtImplemented(in SOMTMethodEntryC entry);
```

Returns 1 (true) if <entry> is a method introduced or overridden by the target class and its visibility matches <somtTargetType> (somtImplementationE matches both private and public)

somtOverridden method

```
boolean somtOverridden(in SOMTMethodEntryC entry);
```

Returns 1 (true) if <entry> is an overridding method of the target class and its visibility matches <somtTargetType> (somtImplementationE matches both private and public)

somtInherited method

```
boolean somtInherited(in SOMTMethodEntryC entry);
```

Returns 1 (true) if <entry> is inherited by the target class and its visibility matches <somtTargetType> (somtImplementationE matches both private and public)

somtAllVisible method

```
boolean somtAllVisible(in SOMTMethodEntryC entry);
```

Returns 1 (true) if <entry> is supported by the target class and its visibility matches <somtTargetType> (somtImplementationE matches both private and public)

somtAll method

```
boolean somtAll(in SOMTMethodEntryC entry);
```

Returns 1 (true) if <entry> is supported by the target class.

somtNewNoProc method

```
boolean somtNewNoProc(in SOMTEntryC entry);
```

Returns 1 (true) if somtNew does and the method IS NOT a direct call Procedure.

somtPrivOrPub method

```
boolean somtPrivOrPub(in SOMTEntryC entry);
```

Returns 1 (true) if entry is Private or Public.

somtNewProc method

```
boolean somtNewProc(in SOMTEntryC entry);
```

Returns 1 (true) if somtNew does and the method IS a direct call Procedure.

somtLink method

```
boolean somtLink(in SOMTEntryC entry);
```

Returns 1 (true) if “nolink” is not set.

somtVA method

```
boolean somtVA(in SOMTEntryC entry);
```

Returns 1 (true) if entry is a VarArgs method.

somtScanMethods method

```
boolean somtScanMethods(in string filter, in string prolog, in string each,
in string epilog, in boolean forceProlog);
```

Will only call <each> on methods accepted by <filter>. If <forceProlog> is not true then the prolog and epilog emiters will be called only if there is at least one method that passes the filter.

somtScanConstants method

```
boolean somtScanConstants(in string prolog, in string each, in string
epilog);
```

somtScanTypedefs method

```
boolean somtScanTypedefs(in string prolog, in string each, in string
epilog);
```

somtScanStructs method

```
boolean somtScanStructs(in string prolog, in string each, in string epilog);
```

somtScanUnions method

```
boolean somtScanUnions(in string prolog, in string each, in string epilog);
```

somtScanEnums method

```
boolean somtScanEnums(in string prolog, in string each, in string epilog);
```

somtScanData method

```
boolean somtScanData(in string prolog, in string each, in string epilog);
```

somtScanAttributes method

```
boolean somtScanAttributes(in string prolog, in string each, in string
epilog);
```

somtScanInterfaces method

```
boolean somtScanInterfaces(in string prolog, in string each, in string
epilog);
```

somtScanModules method

```
boolean somtScanModules(in string prolog, in string each, in string epilog);
```

somtScanPassthru method

```
boolean somtScanPassthru(in boolean before, in string prolog, in string
each, in string epilog);
```

somtEmitFullPassthru method

```
void somtEmitFullPassthru(in boolean before, in string language);
```

Emits each passthru section defined for the language and targetType, and the result of the somtIsBeforePassthru method is equal to the before parameter. (before = 1(true), or before = 0(false), i.e. after.)

somtScanDataF method

```
boolean somtScanDataF(in string filter, in string prolog, in string each, in
string epilog, in boolean forceProlog);
```

This method is like somtScanData but it also provides a paramater for a filter method.

somtScanBasesF method

```
boolean somtScanBasesF(in string filter, in string prolog, in string each,
in string epilog, in boolean forceProlog);
```

This method is like somtScanBases but it also provides a paramater for a filter method.

somtGetGlobalModifierValue method

```
string somtGetGlobalModifierValue(in string modifierName);
```

Returns the value of the specified global modifier.

Global modifiers are specified when the SOM Compiler is invoked, via the “-m” option. For example,

```
sc -m"foo=bar"  file.idl
```

specifies to the SOM Compiler and the emitters being run that the global modifier “foo” has the value “bar.”

Values of global modifiers are transient; they last only for the duration of the compile for which they were specified.

If a modifier is specified in the “sc” command with no value, as in

```
sc -mfoo file.idl
```

then the result of this method will be non-NULL.

If no such modifier is specified, then the result is NULL.

Older SOM compiler version uses “-a” option which is same as “-m” option.

somtGetFirstGlobalDefinition method

```
SOMTEntryC somtGetFirstGlobalDefinition();
```

Returns the first type or constant definition that is not associated with any interface or module.

These global definitions must be surrounded by the somemittypes pragmas for them to be visible via this method. E.g.,

```
#pragma somemittypes on
....
#pragma someemittypes off
```

The list of global definitions returned by this method and the somtGetNextGlobalDefinition method may include entries for forward declarations as well as typedefs and constants.

Global structs and unions are also included in the list.

somtGetNextGlobalDefinition method

```
SOMTEntryC somtGetNextGlobalDefinition();
```

Returns the next type or constant definition that is not associated with any interface or module, relative to a previous call to somtGetFirstGlobalDefinition or somtGetNextGlobalDefinition.

SOMTEntryC Class

somtEntryName attribute

```
attribute string somtEntryName;
```

The name associated with this entry. Eg, the name of the data item, the class, the method, the type, etc.

somtElementType attribute

```
attribute SOMTTypes somtElementType;
```

Returns the type of this entry. This is not datatype, but entry type (method, class, passthru, etc.). The value is defined by SOMTTypes.

somtElementTypeName attribute

```
readonly attribute string somtElementTypeName;
```

String version of somtElementType.

somtEntryComment attribute

```
readonly attribute string somtEntryComment;
```

Returns the comment associated with this entry, or NULL is this entry has no associated comment. Comments will have comment delimitors removed, but will retain newline characters as specified in the source file. (use smLookupComment)

somtSourceLineNumber attribute

```
readonly attribute unsigned long somtSourceLineNumber;
```

Returns the line number in the source file where this entry's syntactic form ended.

somtTypeCode attribute

```
readonly attribute TypeCode somtTypeCode;
```

The typecode, if appropriate, or NULL.

somtIsReference attribute

```
readonly attribute boolean somtIsReference;
```

Whether the entry is just a reference to the real type (TRUE) rather than a declaration of it (FALSE).

somtIDLScopedName attribute

```
readonly attribute string somtIDLScopedName;
```

The IDL scoped name of the entry (using double colon as delimiter).

somtCScopedName attribute

```
readonly attribute string somtCScopedName;
```

The C scoped name of the entry (using underscore as delimiter).

somtGetModifierValue method

```
string somtGetModifierValue(in string modifierName);
```

Returns the value of the named modifier if this entry has the named modifier and NULL otherwise. Note: if the modifier is present but does not have a value then a value of <'\1'> is returned.

somtGetFirstModifier method

```
boolean somtGetFirstModifier(inout string modifierName, inout string
modifierValue);
```

Returns the first modifier associated with this entry. 1 (true) is returned if the entry has at least one modifier and 0 (false) otherwise.

somtGetNextModifier method

```
boolean somtGetNextModifier(inout string modifierName, inout string
modifierValue);
```

Returns the next modifier (with respect to the last call to <somtGetNextModifier> or <somtGetFirstModifier>) associated with this entry. 1 (true) is returned if the entry had another modifier and 0 (false) otherwise.

somtFormatModifier method

```
long somtFormatModifier(in string buffer, in string name, in string value);
```

Formats the indicated name/value pair into buffer. Buffer must be big enough to hold all the formatted pair, no checks are made. The number of characters added to buffer are returned (not including the trailing null character).

Note: value may be null

You will probably never call this method, it is provided so that you can override it to control the format returned in <somtGetModifierList>.

somtGetModifierList method

```
long somtGetModifierList(in string buffer);
```

The modifiers for this entry are placed in <buffer> in template list form (newline separated). Buffer must be big enough to hold all the modifiers, no checks are made. The number of modifiers is returned.

somtSetSymbolsOnEntry method

```
long somtSetSymbolsOnEntry(in SOMTEmitC emitter, in string prefix);
```

Places a number of symbol/value pairs in <t>. All the symbols will begin with <prefix>.

somtSetEntryStruct method

```
void somtSetEntryStruct(inout Entry es);
```

Sets the entry struct data member.

Note, when overridding this method, it is important to call the parent version of the method first and then do your processing.

SOMTEnumEntryC Class

somtGetFirstEnumName method

```
SOMTEnumNameEntryC somtGetFirstEnumName();
```

somtGetNextEnumName method

```
SOMTEnumNameEntryC somtGetNextEnumName();
```

SOMTEnumNameEntryC Class

somtEnumPtr attribute

```
readonly attribute SOMTEnumEntryC somtEnumPtr;
```

A pointer to the enumerator.

somtEnumVal attribute

```
readonly attribute unsigned long somtEnumVal;
```

The value of the enumeration.

SOMTMetaClassEntryC Class

somtMetaFile attribute

```
readonly attribute string somtMetaFile;
```

Returns the name of the file containing the definition of the meta class named in this entry.

somtMetaClassDef attribute

```
readonly attribute SOMTClassEntryC somtMetaClassDef;
```

Returns the class definition entry for the meta class named in this entry.

SOMTMethodEntryC Class

somtIsVarargs attribute

```
readonly attribute boolean somtIsVarargs;
```

Returns 1 (true) if this method definition has a variable length parameter list.

somtOriginalMethod attribute

```
readonly attribute SOMTMethodEntryC somtOriginalMethod;
```

If this is an override method definition (<SOMTOverrideMethodE>) then this is the method definition entry that orginially introduced the method.

somtOriginalClass attribute

```
readonly attribute SOMTClassEntryC somtOriginalClass;
```

If this is an override method definition (<SOMTOverrideMethodE>) then this is the class definition entry that orginially introduced the method.

somtMethodGroup attribute

```
readonly attribute SOMTEntryC somtMethodGroup;
```

The group this method is defined in within a class definition.

somtIsPrivateMethod attribute

```
readonly attribute boolean somtIsPrivateMethod;
```

Whether or not the method is private.

somtIsOneway attribute

```
readonly attribute boolean somtIsOneway;
```

Whether or not the method is oneway.

somtArgCount attribute

```
readonly attribute short somtArgCount;
```

The number of arguments for the method.

somtGetFirstParameter method

```
SOMTParameterEntryC somtGetFirstParameter();
```

Returns the first formal parameter entry for this method if it has one and NULL otherwise. Note: the target object parameter is not included, therefore the first parameter is really the second parameter from a SOM runtime perspective.

somtGetNextParameter method

```
SOMTParameterEntryC somtGetNextParameter();
```

Returns the next formal parameter entry for this method if it has one and NULL otherwise.

somtGetIDLParamList method

```
string somtGetIDLParamList(in string buffer);
```

Returns the formal parameter list (in IDL syntax) for this method. The parameter list is built in <buffer> and the address of <buffer> is returned.

Parameters are delimited with newlines.

The method receiver and any implicit method arguments are NOT included.

somtGetShortCParamList method

```
string somtGetShortCParamList(in string buffer, in string selfParm, in
string varargsParm);
```

Returns the formal parameter list (in ANSI C function prototype form, with types) for this method. The parameter list is built in <buffer> and the address of <buffer> is returned.

Parameters are delimited with newlines.

If this method takes a variable number of arguments then the final parameter substring is replaced by <varargsParm>, unless <varargsParm> is NULL in which case the final parameter is removed.

If <selfParm> is not null then it is added as an initial parameter. (The <selfParm> string may actually contain multiple parameters, delimited by newline characters.)

The method receiver and any implicit method arguments are NOT included.

The types of the method parameters are given in C form (with pointer stars, where needed) rather than in the IDL form.

somtGetFullCParamList method

```
string somtGetFullCParamList(in string buffer, in string varargsParm);
```

Same as somtGetShortCParamList except that the method receiver and any implicit method arguments (Environment and Context) are included. The types of the method parameters are given in C form (with pointer stars, where needed) rather than in the IDL form.

somtGetShortParamNameList mwthod

```
string somtGetShortParamNameList(in string buffer, in string selfParm, in
string varargsParm);
```

Returns the parameter list for this method in call form (without types). The argument list is built in <buffer> and the address of <buffer> is returned. Parameters are delimited with newlines.

If this method takes a variable number of arguments then the final parameter is replaced by <varargsParm>, unless <varargsParm> is NULL in which case the final parameter is removed.

If <selfParm> is not null then it is added as an initial parameter. (The <selfParm> string may actually contain multiple parameters, delimited by newline characters.)

The method receiver and any implicit method arguments are NOT included.

somtGetFullParamNameList method

```
string somtGetFullParamNameList(in string buffer, in string varargsParm);
```

Same as somtGetParamNameList except that the method receiver and any implicit method arguments (Environment and Context) are included.

somtGetNthParameter mwthod

```
SOMTParameterEntryC somtGetNthParameter(in short n);
```

Returns the object representing the nth explicit method parameter.

somtGetFirstException method

```
SOMTStructEntryC somtGetFirstException();
```

The first exception this method raises.

somtGetNextException method

```
SOMTStructEntryC somtGetNextException();
```

The next exception this method raises, relative to the previous call to this method or to somtGetFirstException.

somtContextArray attribute

```
readonly attribute string *somtContextArray;
```

An array of the context string-literals for the method.

somtCReturnType attribute

```
readonly attribute string somtCReturnType;
```

The C datatype the method returns. This may not correspond to the IDL data type (in particular, pointer stars may be added).

SOMTModuleEntryC Class

somtOuterModule attribute

```
readonly attribute SOMTModuleEntryC somtOuterModule;
```

The module enclosing this module, or NULL if there is none.

somtModuleFile attribute

```
readonly attribute string somtModuleFile;
```

The name of the file in which the module appears.

somtGetFirstModuleStruct method

```
SOMTStructEntryC somtGetFirstModuleStruct();
```

somtGetNextModuleStruct method

```
SOMTStructEntryC somtGetNextModuleStruct();
```

somtGetFirstModuleTypedef method

```
SOMTTypedefEntryC somtGetFirstModuleTypedef();
```

somtGetNextModuleTypedef method

```
SOMTTypedefEntryC somtGetNextModuleTypedef();
```

somtGetFirstModuleUnion method

```
SOMTUnionEntryC somtGetFirstModuleUnion();
```

somtGetNextModuleUnion method

```
SOMTUnionEntryC somtGetNextModuleUnion();
```

somtGetFirstModuleEnum method

```
SOMTEnumEntryC somtGetFirstModuleEnum();
```

somtGetNextModuleEnum method

```
SOMTEnumEntryC somtGetNextModuleEnum();
```

somtGetFirstModuleConstant mwthod

```
SOMTConstEntryC somtGetFirstModuleConstant();
```

somtGetNextModuleConstant mwthod

```
SOMTConstEntryC somtGetNextModuleConstant();
```

somtGetFirstModuleSequence method

```
SOMTSequenceEntryC somtGetFirstModuleSequence();
```

somtGetNextModuleSequence method

```
SOMTSequenceEntryC somtGetNextModuleSequence();
```

somtGetFirstInterface method

```
SOMTClassEntryC somtGetFirstInterface();
```

somtGetNextInterface method

```
SOMTClassEntryC somtGetNextInterface();
```

somtGetFirstModule method

```
SOMTModuleEntryC somtGetFirstModule();
```

somtGetNextModule method

```
SOMTModuleEntryC somtGetNextModule();
```

somtGetFirstModuleDef method

```
SOMTEntryC somtGetFirstModuleDef();
```

somtGetNextModuleDef method

```
SOMTEntryC somtGetNextModuleDef();
```

SOMTParameterEntryC Class

somtParameterDirection attribute

```
readonly attribute somtParameterDirectionT somtParameterDirection;
```

The direction for this parameter. (somtInE, somtOutE, or somtInOutE).

somtIDLParameterDeclaration attribute

```
readonly attribute string somtIDLParameterDeclaration;
```

The IDL declaration of the parameter, including the type and name.

somtCParameterDeclaration attribute

```
readonly attribute string somtCParameterDeclaration;
```

The declaration for the parameter within a C method procedure prototype. It includes the parameter's type and name. This may differ from the parameter's IDL declaration. In particular, pointer stars may be added.

somtPascalParameterDeclaration attribute

```
readonly attribute string somtPascalParameterDeclaration;
```

The declaration for the parameter within a Pascal method procedure prototype. It includes the parameter's type and name. This may differ from the parameter's IDL declaration. In particular, pointer stars may be added.

SOMTPassthruEntryC Class

somtPassthruBody attribute

```
readonly attribute string somtPassthruBody;
```

The source content text of this passthru entry without any modification. Newlines that were present in the source will still be present.

somtPassthruLanguage attribute

```
readonly attribute string somtPassthruLanguage;
```

Returns the name of the language for which this passthru entry is intended. Language names are always all upper case.

somtPassthruTarget attribute

```
readonly attribute string somtPassthruTarget;
```

Returns the target for this passthru entry.

somtIsBeforePassthru method

```
boolean somtIsBeforePassthru();
```

Returns 1 (true) if this passthru entry is to be put at the beginning of the file or 0 (false) if this passthru entry is to go later in the file.

SOMTSequenceEntryC Class

somtSeqLength attribute

```
readonly attribute long somtSeqLength;
```

The length of the sequence.

somtSeqType attribute

```
readonly attribute SOMTEntryC somtSeqType;
```

The type of the sequence.

SOMTStringEntryC Class

somtStringLength attribute

```
readonly attribute long somtStringLength;
```

The length of the string.

SOMTStructEntryC Class

somtGetFirstMember Method

```
SOMTTypedefEntryC somtGetFirstMember();
```

The first member of the struct.

somtGetNextMember Method

```
SOMTTypedefEntryC somtGetNextMember();
```

The next member of the struct, relative to the previous call to this method or somtGetFirstMember.

somtStructClass method

```
readonly attribute SOMTClassEntryC somtStructClass;
```

The class in which the structure was defined.

somtIsException method

```
readonly attribute boolean somtIsException;
```

Whether the structure is really an exception.

SOMTTemplateOutputC Class

somtAddSectionDefinitions Method

```
void somtAddSectionDefinitions(in string defString);
```

Add section definitions from <defString> buffer to Symbol table.

somtCommentStyle attribute

```
attribute somtCommentStyleT somtCommentStyle;
```

Set style of output comment. Supported styles are:

- somtDashesE: “–” at the start of each line
- somtCPPE: C++ style, "//" at the start of each line
- somtCSimpleE: simple C style, each line wrapped in /* and */
- somtCBlockE: block C style, block style, ie leading /* then a * on each line and then a final */
- somtPSimpleE: simple Pascal style, each line wrapped in (* and *)
- somtPBlockE: block Pascal style, block style, ie leading (* then a * on each line and then a final *)
- somtPBorlandE: block Borland Pascal style, block style, ie leading { and then a final }

somtLineLength attribute

```
attribute long somtLineLength;
```

Line length limit. At least on list item will be output.

somtCommentNewline attribute

```
attribute boolean somtCommentNewline;
```

Output comment block from new line flag.

somtCheckSymbol Method

```
boolean somtCheckSymbol(in string name);
```

Return TRUE id symbol <name> exists in Symbol Table.

somtExpandSymbol Method

```
string somtExpandSymbol(in string s, in string buf);
```

somtGetSymbol Method

```
string somtGetSymbol(in string name);
```

Return symbol value for <name> from Symbol table.

somto Method

```
void somto(in string tmplt);
```

Outputs a template, <tmplt>, after substitution for any symbols that occur in it. Five substitutions are supported: simple, list, comment, tab, and conditional.

Substitutable items in the template are bracketed with angle brackets. (Backslash can be used to escape an angle bracket.)

Simple substitutions just replace a symbol with its value. If the symbol has no value in this template object then the symbol is replaced error string but no error is raised.

List substitution assumes that the symbol has a value in output template list form. This is a newline separated string of values. The list substitution specification consists of four parts, a prefix, a symbol, a separator, and a list indicator. prefixes and separators can only be composed of blanks, comma, colons, and semi-colons. The list indicator is "..." (three periods). For example, the list substitution specification "<, name, ...> has a prefix of ", ", a symbol of "name" and a separator of ", ". The prefix will be used whenever there is at least one item in the list and the separator will be used between any two list items. After the first items of a list is placed each additional item is evaluated to see if it would begin after the line length limit (set by _set_somtLineLength), if it would then a new line is begun and the value is placed directly under the first item. Comment substitution assumes that the symbol has a value in output template list form. A comment specification consists of a comment indicator followed by a symbol name. The comment indicator is "–". Eg, <-- classComment> is a valid comment substitution specification. The lines of the comment are output according to the current comment style (see <somtCommentStyle>) and aligned with the starting column of the comment specification. Tab substitution is specified by <@dd> where "dd" is a valid positive integer. Blanks will be inserted into the output stream if necessary to position the next character of output at the column indicated by "dd".

Conditional substitution is specified by putting a question mark, "?", in column one of the template line. The line will not be output at all unless at least one valid, non-blank, symbol substitution occurs on the line.

Note: Due design error in IBM SOM 3.0 this method can't be fully replaced. You can do some preprocessing of <templ> and call parent method. This is due direct usage of FILE structure in somto method. This means you can't write to file using standard C file functions because FILE structure is a compiler depended. But you don't know which compiler was used for. Header files contains compiler-

independed file functions (somtok*), but no any of this functions, except two ones, exported in SOM DLLs. So, if you want to fully replace this method then you need also replace lot of other methods and functions of Emitter Framework and SOM Compiler library. For IBM SOM 2.1 all seems to be ok, but you must use somtok* functions from SOMC.DLL, not standard C runtime for file operations.

somtOutputComment Method

```
void somtOutputComment(in string comment);
```

Outputs comment using comment style settings.

Note: Due design error in IBM SOM 3.0 this method can't be fully replaced. You can do some preprocessing of <comment> and call parent or somto method. This is due direct usage of FILE structure in somto method. This means you can't write to file using standard C file functions because FILE structure is a compiler depended. But you don't know which compiler was used for. Header files contains compiler-independed file functions (somtok*), but no any of this functions, except two ones, exported in SOM DLLs. So, if you want to fully replace this method then you need also replace lot of other methods and functions of Emitter Framework and SOM Compiler library. For IBM SOM 2.1 all seems to be ok, but you must use somtok* functions from SOMC.DLL, not standard C runtime for file operations.

somtOutputSection Method

```
void somtOutputSection(in string sectionName);
```

Same as somto method, but template read from Symbol table with key equal to sectionName. Uses somto method for actual output.

somtReadSectionDefinitions Method

```
void somtReadSectionDefinitions(inout FILE fp);
```

This method reads sections from template file and stores them in Symbol table. fp is a value returned by somtOpenSymbolsFile method of SOMTEmitC class.

Note: Due design error in IBM SOM 3.0 this method can't be replaced. This is due unknown structure of FILE type. This means you can't read file using standard C file functions because FILE structure is a compiler depended. But you don't know which compiler was used for. Header files contains compiler-independed file functions (somtok*), but no any of this functions, except two ones, exported in SOM DLLs. So, if you want to fully replace this method then you need also replace lot of other methods and functions of Emitter Framework and SOM Compiler library. For IBM SOM 2.1 all seems to be ok, but you must use somtok* functions from SOMC.DLL, not standard C runtime for file operations.

somtSetOutputFile Method

```
void somtSetOutputFile(inout FILE fp);
```

Pass FILE structure to object to use for file I/O. fp is a value returned by somtOpenEmitFile or somtOpenEmitFileSL.

Note: FILE structure must be same as in other I/O methods and functions.

somtSetSymbol Method

```
void somtSetSymbol(in string name, in string value);
```

Set symbol name in Symbol table to value. name and value must be allocated using SOMMaloc function. It will be deallocated using SOMFree on object destroying.

somtSetSymbolCopyBoth Method

```
void somtSetSymbolCopyBoth(in string name, in string value);
```

Same as somtSetSymbol but name and value will be copied to internally allocated buffer.

somtSetSymbolCopyName Method

```
void somtSetSymbolCopyName(in string name, in string value);
```

Same as somtSetSymbol but name will be copied to internally allocated buffer.

somtSetSymbolCopyValue Method

```
void somtSetSymbolCopyValue(in string name, in string value);
```

Same as somtSetSymbol but value will be copied to internally allocated buffer.

SOMTTypedefEntryC Class

somtTypedefType attribute

```
readonly attribute SOMTEntryC somtTypedefType;
```

The type of the typedef. This does not include pointer stars or array declarators. These must be obtained by examining each of the declarators.

somtGetFirstDeclarator method

```
SOMTCommonEntryC somtGetFirstDeclarator();
```

The first declarator for this typedef. Declarators of struct members will be instances of SOMTDataEntryC, while declarators of typedefs will be instances of SOMTUserDefinedTypeEntryC.

somtGetNextDeclarator method

```
SOMTCommonEntryC somtGetNextDeclarator();
```

The next declarator for this typedef, relative to the previous call to this method or somtGetFirstDeclarator. Declarators of struct members will be instances of SOMTDataEntryC, while declarators of typedefs will be instances of SOMTUserDefinedTypeEntryC.

SOMTUnionEntryC Class

somtSwitchType attribute

```
readonly attribute SOMTEntryC somtSwitchType;
```

The switch type of the union.

somtGetFirstCaseEntry method

```
somtCaseEntry *somtGetFirstCaseEntry();
```

The first case for the union.

somtGetNextCaseEntry method

```
somtCaseEntry *somtGetNextCaseEntry();
```

The next case for the union, relative to the previous call to this method or to somtGetFirstCaseEntry.

SOMTUserDefinedTypeEntryC Class

somtOriginalTypedef attribute

```
readonly attribute SOMTTypedefEntryC somtOriginalTypedef;
```

The typedef that defined the user-defined type.

somtBaseTypeObj attribute

```
readonly attribute SOMTEntryC somtBaseTypeObj;
```

The object representing the base type (eg. short, float, unsigned long) of a user-defined type, skipping over any intermediate user-defined types.

SOMStringTableC Class

```
interface SOMStringTableC : SOMObject
```

Объектами класса SOMStringTableC являются символьные таблицы, которые отображают строки на строки (ключ-значение, ассоциативные массивы). Любой экземпляр класса может хранить неограниченное число элементов. При увеличении количества строк время поиска строки увеличивается. В отличие от IBM SOM в данной реализации не используются хэш-таблицы.

somstTargetCapacity attribute

```
attribute unsigned long somstTargetCapacity;
```

Емкость ассоциативного массива. Значение не влияет на работу и сохранено для совместимости. В IBM SOM данный атрибут определял размер хэш-таблицы. Данный атрибут должен выставляться до вызова любого из методов данного класса

somstAssociationsCount attribute

```
readonly attribute unsigned long somstAssociationsCount;
```

Текущее число ассоциаций в массиве

somstAssociate method

```
short somstAssociate(in string key, in string value);
```

Устанавливает связь <key> и <value>. Возвращает 0, если связь не может быть установлена (<key> нулевой или недостаточно памяти); -1 - ассоциация успешна выполнена, но <key> уже имел значение до вызова метода, 1 - ассоциация успешно выполнена и <key> не существовал. Замечание: массив сохраняет ссылки на <key> и <value>, передаваемые в аргументах. Копия значений <key> и <value> не создается. При уничтожении объекта память, занимаемая <key> и <value> освобождается с помощью SOMFree, т.е. память под <key> и <value> должна быть выделена с помощью SOMMalloc и аналогичных функций. Замечание: При замене <value> при имеющемся <key> старое <value> заменяется, память не освобождается

somstAssociateCopyKey method

```
short somstAssociateCopyKey(in string key, in string value);
```

То же, что и <somstAssociate>, но массив содержит копии значений <key>. Значение <key> копируется в выделяемую с помощью SOMMalloc память.

somstAssociateCopyValue method

```
short somstAssociateCopyValue(in string key, in string value);
```

То же, что и <somstAssociate>, но массив содержит копии значений <value>. Значение <value> копируется в выделяемую с помощью SOMMalloc память.

somstAssociateCopyBoth method

```
short somstAssociateCopyBoth(in string key, in string value);
```

То же, что и <somstAssociate>, но массив содержит копии значений <key> и <value>. Значения <key> и <value> копируются в выделяемую с помощью SOMMalloc память.

somstGetAssociation method

```
string somstGetAssociation(in string key);
```

Возвращается строка, ассоциированная с <key>, или NULL, если нет ассоциации. Массив продолжает хранить указатель на значение.

somstClearAssociation method

```
boolean somstClearAssociation(in string key);
```

The association for <key>, if any, is removed.1 is returned if <key> had an association, and 0 is returned if it did not.

somstGetIthKey method

```
string somstGetIthKey(in unsigned long i);
```

Возвращает ключевую часть <i>-й по счету ассоциации. Если нет ассоциации, то возвращает NULL. Порядок ассоциации в массиве не определен, но остается постоянным до следующей модификации.

somstGetIthValue method

```
string somstGetIthValue(in unsigned long i);
```

Возвращает значимую часть <i>-й по счету ассоциации. Если нет ассоциации, то возвращает NULL. Порядок ассоциации в массиве не определен, но остается постоянным до следующей модификации.

somtStrDup function

```
SOMEXTERN char * SOMLINK somtStrDup(char *str);
```

Allocate memory and duplicate string str

somtEntryTypeName function

```
SOMEXTERN char * SOMLINK somtEntryTypeName(SOMTTypes type);
```

Return string representation of type of Entry structure except special case SOMTEmitterBeginE and SOMTEmitterEndE types.

somtShowEntry function

```
SOMEXTERN void SOMLINK somtShowEntry(Entry * ep);
```

Output using somPrintf information about Entry structure.

somtStrCat function

```
SOMEXTERN char * SOMLINK somtStrCat(int count,...);
```

Concatenate count of strings.

somtMakeIncludeStr function

```
SOMEXTERN char * SOMLINK somtMakeIncludeStr(boolean local, char *stem, char
*suffix);
```

Produce include string for local (include ”“) or global (include <>) using file stem as file name and suffix as file extension.

somtNewSymbol function

```
SOMEXTERN char * SOMLINK somtNewSymbol(char *prefix, char *stem);
```

Allocate memory and produce string from prefix and stem.

somtGetFileStem function

```
SOMEXTERN char * SOMLINK somtGetFileStem(char *fullName);
```

Allocate memory and return file stem from file name.

somtGetObjectWrapper function

```
SOMEXTERN SOMTEntryC * SOMLINK somtGetObjectWrapper(Entry * ep);
```

Return SOMT*EntryC object for ep Entry structure.

Mapping of Entry types to SOMT*EntryC classes:

Entry type	**Emitter Framework Class**
SOMTArgumentE	SOMTParameterEntryC
SOMTAttE	SOMTAttributeEntryC
SOMTBadEntryE	Fatal error
SOMTBaseE	SOMTBaseClassEntryC
SOMTClassE	SOMTClassEntryC
SOMTConstE	SOMTConstEntryC
SOMTDataE	SOMTDataEntryC
SOMTEnumBE	SOMTEnumNameEntryC
SOMTEnumE	SOMTEnumEntryC
SOMTEnumPE	SOMTEnumEntryC
SOMTFloatBE	SOMTEntryC
SOMTAnyBE	SOMTEntryC
SOMTGroupE	SOMTEntryC
SOMTCopyrightE	SOMTEntryC
SOMTLongBE	SOMTEntryC
SOMTNegativeBE	SOMTEntryC
SOMTOctetBE	SOMTEntryC
SOMTTypeCodeBE	SOMTEntryC
SOMTBooleanBE	SOMTEntryC
SOMTCaseEntryE	SOMTEntryC
SOMTCaseListE	SOMTEntryC
SOMTCaseSTME	SOMTEntryC
SOMTCharBE	SOMTEntryC
SOMTDclListE	SOMTEntryC

Entry type	Emitter Framework Class
SOMTDefaultE	SOMTEntryC
SOMTDoubleBE	SOMTEntryC
SOMTEBaseE	SOMTEntryC
SOMTEEnumE	SOMTEntryC
SOMTShortBE	SOMTEntryC
SOMTStringBE	SOMTEntryC
SOMTUnsignedLongBE	SOMTEntryC
SOMTUnsignedShortBE	SOMTEntryC
SOMTVoidBE	SOMTEntryC
SOMTVoidPtrBE	SOMTEntryC
SOMTMetaE	SOMTMetaClassEntryC
SOMTModuleE	SOMTModuleEntryC
SOMTNewMethodE	SOMTMethodEntryC
SOMTOverriddenMethodE	SOMTMethodEntryC
SOMTOverrideMethodE	SOMTMethodEntryC
SOMTPassthruE	SOMTPassthruEntryC
SOMTSequenceE	SOMTSequenceEntryC
SOMTSequenceTDE	SOMTSequenceEntryC
SOMTStringE	SOMTStringEntryC
SOMTStructE	SOMTStructEntryC
SOMTStructPE	SOMTStructEntryC
SOMTStructSE	SOMTStructEntryC
SOMTTyDclE	SOMTTypedefEntryC
SOMTTypedefE	SOMTTypedefEntryC
SOMTTypedefBE	SOMTUserDefinedTypeEntryC
SOMTUnionE	SOMTUnionEntryC
SOMTUnionPE	SOMTUnionEntryC
SOMTUnionSE	SOMTUnionEntryC
SOMTEmitterBeginE	Fatal error
SOMTEmitterEndE	Fatal error

Appendixes

1. Appendix 1. SOM ABI

Due switching from MSVC (IBM SOM 2.1) to VAC (IBM SOM 3.0) some problems was occur:

First problem is a calling convention. All non SOMLINK calls in IBM SOM 2.1 is a _cdecl calls. But under IBM SOM 3.0 all non SOMLINK calls is a Optlink calls. Read some info here: https://github.com/prokushev/SOM-Delphi-Wiki/blob/master/Known%20differences%20between%20SOM%202.1%20and%20SOM%203.0.md Goal of somFree SOM Compiler and Emitter Framework is to provide a possibility to use original IBM SOM emitters as from IBM SOM 2.1 as from IBM SOM 3.0. Another goal is a development of somFree emitters, which can be used on both IBM SOM 2.1 and IBM SOM 3.0 compilers.To achieve above goals somFree provides some solutions: 1. Automatic somc.dll

calling convention switching. somFree SOMC.DLL provides automatic switching of IBM SOM 2.1 ABI and IBM SOM 3.0 ABI. Switching occurs on somtload call during loading of emitter. For IBM SOM 3.0 all emitter contains entry point emitSL, so, if loading was success, then somFree handles Optlink calling convention for all non SOMLINK calls. If no such entry (found only emit) then IBM SOM 2.1 ABI used. 2. Support both entry points (emitSL and emit) in emitters. somFree emitters automatically switches to IBM SOM 2.1 ABI on emit call and to IBM SOM 3.0 ABI on emitSL call.

Список литературы

1. Object Management Group, «C Language Mapping Specification 1.0,» [В Интернете]. Available: https://www.omg.org/spec/C/. [Дата обращения: 24 Август 2022].
2. IBM, OS/2 2.0 Technical Library. System Object Model Guide and Reference. First Edition., 1991. https://www.os2museum.com/files/docs/os220tl/os2-2.0-som-1991.pdf