Reference for Emitter Framework Classes and Methods

SOMTAttributeEntryC Class

Description

A **SOMTAttributeEntryC** object represents an attribute declaration statement in a class interface definition. It provides attributes for accessing the type of the attribute and whether it is readonly, and methods for accessing the names of the attributes being declared and their get/set methods.

File Stem

scattrib

Base

SOMTEntryC

Metaclass

SOMClass

Ancestor Classes

SOMTEntryC, **SOMObject**

Attributes

Listed below is each available attribute, with its corresponding type in parentheses, followed by a description of its purpose:

somtIsReadonly (boolean)
Whether the attribute is defined as readonly.
This attribute has no “set” method.

somtAttribType (SOMTEntryC)
A pointer to an entry object representing the base type of the attribute.
This does not include pointer stars or array declarators; to get the full type, get each attribute declarator in turn and get its **somtType** attribute.
This attribute has no “set” method.

New Methods

somtGetFirstAttributeDeclarator
somtGetNextAttributeDeclarator
somtGetFirstGetMethod
somtGetNextGetMethod
somtGetFirstSetMethod
somtGetNextSetMethod

Overriding Methods

somtSetSymbolsOnEntry
somDumpSelfInt

somtGetFirst<*Item*> Methods

Purpose

These methods get the first declarator, “get” method, or “set” method for an attribute entry.

IDL Syntax

SOMTDataEntryC somtGetFirstAttributeDeclarator ( );

SOMTMethodEntryC somtGetFirstGetMethod ( );

SOMTMethodEntryC somtGetFirstSetMethod ( );

Note: These methods do *not* take an **Environment** parameter.

Description

The **somtGetFirst**<*Item*> methods return the first item of the type shown above for the entry specified by *receiver*. The next item of the same kind can be obtained using the corresponding **somtGetNext**<*Item*> method. For example, the **somtGetFirstAttributeDeclarator** method returns the entry representing the first declarator of the specified attribute entry. The **somtGetNextAttributeDeclarator** can be used repeatedly to retrieve each successive declarator.

Note that the same **somtGetFirst**<*Item*> and **somtGetNext**<*Item*> methods cannot be used in doubly nested loops. For example, the following doubly nested loop will not work, because after the first execution of the inner loop, the invocation of **somtGetNextAttributeDeclarator** in the outer loop will return NULL:

```
for (d1 = _somtGetFirstAttributeDeclarator(attrib); d1;
     d1 = _somtGetNextAttributeDeclarator(attrib))
   for (d2 = _somtGetFirstAttributeDeclarator(attrib); d2;
        d2 = _somtGetNextAttributeDeclarator(attrib))
    /* etc. */
```

Nested loops such as the one above are permissible if the target object (for example, “attrib”) of the inner loop differs from the target object of the outer loop, or if a different **somtGetFirst**<*Item*> method is used in the inner loop (for instance, a **somtGetNextGetMethod** loop can be nested inside a **somtGetNextAttributeDeclarator** loop).

Parameters

receiver The entry whose first item is to be retrieved.

Return Value

These methods return the first declarator, “get” method, or “set” method for an attribute entry.

Example

To iterate through the declarators of an attribute statement:

```
SOMTDataEntryC myEntry;
SOMTAttributeClassEntryC attrib;

printf("List of declarators:\n");
for (myEntry = _somtGetFirstAttributeDeclarator(attrib); myEntry;
    myEntry = _somtGetNextAttributeDeclarator(attrib))
    printf("%s\n", __get_somtEntryName(myEntry));
```

Related Information

Methods: somtGetNext<*Item*>

somtGetNext<*Item*> Methods

Purpose

These methods get the next declarator, “get” method, or “set” method for an attribute entry, relative to the previous call for a similar entry.

IDL Syntax

SOMTDataEntryC somtGetNextAttributeDeclarator ( );

SOMTMethodEntryC somtGetNextGetMethod ( );

SOMTMethodEntryC somtGetNextSetMethod ( );

Note: These methods do *not* take an **Environment** parameter.

Description

The **somtGetNext**<*Item*> methods return the next declarator, “get” method, or “set” method for the entry represented by *receiver*, if it has a next item of that type. Otherwise, it returns NULL.

A call to a **somtGetNext**<*Item*> method is relative to the last call of either the same method or the corresponding **somtGetFirst**<*Item*> method**,** applied to the same entry object.

Note that this implies that the same **somtGetFirst**<*Item*> and **somtGetNext**<*Item*> methods cannot be used in doubly nested loops. For example, the following doubly nested loop will not work, because following the first execution of the inner loop, the invocation of **somtGetNextAttributeDeclarator** in the outer loop will return NULL:

```
for (d1 = _somtGetFirstAttributeDeclarator(attrib); d1;
     d1 = _somtGetNextAttributeDeclarator(attrib))
   for (d2 = _somtGetFirstAttributeDeclarator(attrib); d2;
        d2 = _somtGetNextAttributeDeclarator(attrib))
    /* etc. */
```

Nested loops such as the one above are permissible if the target object (for example, “attrib”) of the inner loop differs from the target object of the outer loop, or if a different **somtGetFirst**<*Item*> method is used in the inner loop (for instance, a **somtGetNextGetMethod** loop can be nested inside a **somtGetNextAttributeDeclarator** loop).

Parameters

receiver The entry whose next item is to be retrieved.

Return Value

These methods return the next item (of the type shown above) for the entry represented by *receiver*, if it has a next item of that type. Otherwise, it returns NULL. The type of item returned is specific to the method; see the method call syntax shown above.

Example

To iterate through the declarators of an attribute statement:

```
SOMTDataEntryC myEntry;
SOMTAttributeClassEntryC attrib;

printf("List of declarators:\n");
for (myEntry = _somtGetFirstAttributeDeclarator(attrib); myEntry;
    myEntry = _somtGetNextAttributeDeclarator(attrib))
    printf("%s\n", __get_somtEntryName(myEntry));
```

Related Information

Methods: somtGetFirst<*Item*>

SOMTBaseClassEntryC Class

Description

A **SOMTBaseClassEntryC** object represents a base class declaration in a class definition. The entry for the base class itself is accessed via the **somtBaseClassDef** attribute.

File Stem

scbase

Base

SOMTEntryC

MetaClass

SOMClass

Ancestor Classes

SOMTEntryC, SOMObject

Attributes

Listed below is each available attribute, with its corresponding type in parentheses, followed by a description of its purpose:

somtBaseClassDef (SOMTClassEntryC)
An entry object representing the definition of the base class named in this entry. This attribute has no “Set” method.

New Methods

None.

Overriding Methods

somDumpSelfInt
somtSetSymbolsOnEntry

SOMTClassEntryC Class

Description

A **SOMTClassEntryC** object represents a complete class interface definition. A **SOMTClassEntryC** object provides methods for accessing the constants, types, structs, unions, enums, sequences, attributes, and methods defined within an interface statement. It also provides methods for accessing the instance data, passthrus, and release names defined in the SOM IDL "implementation" section of the interface statement.

A number of the possible statements in an IDL definition are optional. When they are missing from the class definition, then methods that would return an entry for that kind of statement will return NULL.

File Stem

scclass

Base

SOMTEntryC

Metaclass

SOMClass

Ancestor Classes

SOMTEntryC, **SOMObject**

Attributes

Listed below is each available attribute, with its corresponding type in parentheses, followed by a description of its purpose:

somtSourceFileName (string)
The name of the file containing the class definition.
This attribute has no "Set" method.

somtMetaClassEntry (SOMTMetaClassEntryC)
A pointer to an entry object representing the metaclass statement in a class definition, or NULL if there is none explicitly specified.
This attribute has no "Set" method.

somtClassModule (SOMTModuleEntryC)
The module enclosing this class, or NULL if there is not one.

somtNewMethodCount (long)
The number of methods the class introduces.
This attribute has no "Set" method.

somtStaticMethodCount (long)
The number of static methods the class introduces.
This attribute has no "Set" method.

somtOverrideMethodCount (long)
The number of methods the class overrides.
This attribute has no "Set" method.

somtProcMethodCount (long)
The number of procedure methods the class implements**.**
This attribute has no "Set" method.

somtVAMethodCount (long)
The number of methods in the class that take a variable number of arguments.
This attribute has no “Set” method.

somtBaseCount (int)
The number of base classes for the class.
This attribute has no “Set” method.

somtMetaclassFor (SOMTClassEntryC)
If the class is a metaclass, a pointer to an entry object representing a class for which it is a metaclass.
This attribute has no “Set” method.

somtForwardRef (boolean)
Whether or not this entry represents a forward reference.
This attribute has no “Set” method.

New Methods

Group: scanners

somtGetFirstBaseClass
somtGetNextBaseClass
somtGetFirstReleaseName
somtGetNextReleaseName
somtGetReleaseNameList
somtGetFirstPassthru
somtGetNextPassthru
somtGetFirstData
somtGetNextData
somtGetFirstMethod
somtGetNextMethod
somtGetFirstInheritedMethod
somtGetNextInheritedMethod
somtGetFirstAttribute
somtGetNextAttribute
somtGetFirstConstant
somtGetNextConstant
somtGetFirstStruct
somtGetNextStruct
somtGetFirstUnion
somtGetNextUnion
somtGetFirstEnum
somtGetNextEnum
somtGetFirstTypedef
somtGetNextTypedef
somtGetFirstSequence
somtGetNextSequence
somtGetFirstPubdef
somtGetNextPubdef

Group: filters

somtFilterNew
somtFilterOverridden

Overriding Methods

somDumpSelfInt
somtSetSymbolsOnEntry

somtFilterNew Method

Purpose

Determines whether a method is introduced by a particular class.

IDL Syntax

boolean somtFilterNew (
in SOMTMethodEntryC *method***);**

Note: This method does *not* take an **Environment** parameter.

Description

The **somtFilterNew** method returns TRUE if the specified *method* is introduced by the class represented by the *receiver*. Otherwise, it returns FALSE.

Parameters

receiver An object of class **SOMTClassEntryC** representing a class.

method An object of class **SOMTMethodEntryC** representing the method to be tested.

Return Value

The **somtFilterNew** method returns TRUE if the specified *method* is introduced by the class represented by the *receiver*. Otherwise, it returns FALSE.

Example

```
SOMTClassEntryC cls = __get_somtTargetClass(emitter);
SOMTMethodEntryC method;
method = _somtGetFirstMethod(cls);
if (_somtFilterNew(cls, method))
   printf("Method %s is introduced by %s.\n",
                          __get_somtEntryName(method),
                         __get_somtEntryName(cls));
```

Original Class

SOMTClassEntryC

Related Information

Methods: somtFilterOverridden, somtNew, somtNewProc, somtNewNoProc

somtFilterOverridden Method

Purpose

Determines whether a method is overridden by a particular class.

IDL Syntax

boolean somtFilterOverridden (
in SOMTMethodEntryC *method***);**

Note: This method does *not* take an **Environment** parameter.

Description

The **somtFilterOverridden** method returns TRUE if the specified *method* is overridden by the class represented by the *receiver*. Otherwise, it returns FALSE.

Parameters

receiver An object of class **SOMTClassEntryC** representing a class.

method An object of class **SOMTMethodEntryC** representing the method to be tested.

Return Value

The **somtFilterOverridden** method returns TRUE if the specified *method* is overridden by the class represented by the *receiver*. Otherwise, it returns FALSE.

Example

```
SOMTClassEntryC cls = __get_somtTargetClass(emitter);
SOMTMethodEntryC method;
method = _somtGetFirstMethod(cls);
if (_somtFilterOverridden(cls, method))
   printf("Method %s is an overriding method.\n",
                          __get_somtEntryName(method));
```

Original Class

SOMTClassEntryC

Related Information

Methods: somtFilterNew, somtOverridden

somtGetFirst<*Item*> Methods

Purpose

These methods get the first item (such as a parent class, method, constant, etc.) for a class entry.

IDL Syntax

SOMTAttributeEntryC somtGetFirstAttribute ( );

SOMTBaseClassEntryC somtGetFirstBaseClass ( );

SOMTConstEntryC somtGetFirstConstant ( );

SOMTDataEntryC somtGetFirstData ( );

SOMTEnumEntryC somtGetFirstEnum ( );

SOMTMethodEntryC somtGetFirstInheritedMethod ( );

SOMTMethodEntryC somtGetFirstMethod ( );

SOMTPassthruEntryC somtGetFirstPassthru ( );

string somtGetFirstReleaseName ( );

SOMTSequenceEntryC somtGetFirstSequence ( );

SOMTStructEntryC somtGetFirstStruct ( );

SOMTTypedefEntryC somtGetFirstTypedef ( );

SOMTUnionEntryC somtGetFirstUnion ( );

SOMTEntryC somtGetFirstPubdef ( );

Note: These methods do *not* take an **Environment** parameter.

Description

The **somtGetFirst**<*Item*> methods return the first item of the type shown above for the entry specified by *receiver*, if it has one. Otherwise, it returns NULL. The next item of the same kind can be obtained using the corresponding **somtGetNext**<*Item*> method. For example, the **somtGetFirstMethod** method returns the entry representing the first new or overriding method of the specified class. If the class has no new or overriding methods, it returns NULL. The **somtGetNextMethod** can be used repeatedly to retrieve each successive method. The **somtGetFirstPubdef** method returns the first constant/type definition of the class, whether a typedef, struct, union, etc. If the class does not explicitly declare a metaclass or include the IDL specification for **SOMClass**, then the first "pubdef" will be an entry introducing **SOMClass** as a valid type name.

Note that the same **somtGetFirst**<*Item*> and **somtGetNext**<*Item*> methods cannot be used in doubly nested loops. For example, the following doubly nested loop will not work, because following the first execution of the inner loop, the invocation of **somtGetNextMethod** in the outer loop will return NULL:

```
for (m1 = _somtGetFirstMethod(cls); m1;
     m1 = _somtGetNextMethod(cls))
   for (m2 = _somtGetFirstMethod(cls); m2;
        m2 = _somtGetNextMethod(cls))
    /* etc. */
```

Nested loops such as the one above are permissible if the target object (e.g., "cls") of the inner loop differs from the target object of the outer loop, or if a different **somtGetFirst**<*Item*> method is used in the inner loop (for instance, a **somtGetNextParameter** loop can be nested inside a **somtGetNextMethod** loop).

Parameters

receiver The entry whose first item is to be retrieved.

Return Value

These methods return the first item (such as a parent class, method, constant, etc.) for a class entry. The type of item returned is specific to the method; see the method call syntax shown above.

Example

To iterate through the base classes of a class:

```
SOMTBaseClassEntryC myEntry;

printf("List of base classes:\n");
for (myEntry = _somtGetFirstBaseClass(cls); myEntry;
    myEntry = _somtGetNextBaseClass(cls))
    printf("%s\n", __get_somtEntryName(myEntry));
```

Related Information

Methods: somtGetNext<*Item*>

somtGetNext<*Item*> Methods

Purpose

These methods get the next item (such as a parent class, method, constant, etc.) for a class entry, relative to the previous call for a similar entry.

IDL Syntax

SOMTAttributeEntryC somtGetNextAttribute ( );

SOMTBaseClassEntryC somtGetNextBaseClass ( );

SOMTConstEntryC somtGetNextConstant ( );

SOMTDataEntryC somtGetNextData ( );

SOMTEnumEntryC somtGetNextEnum ( );

somtMethodEntryC somtGetNextInheritedMethod ( );

somtMethodEntryC somtGetNextMethod ( );

SOMTPassthruEntryC somtGetNextPassthru ( );

string somtGetNextReleaseName ( );

SOMTSequenceEntryC somtGetNextSequence ( );

SOMTStructEntryC somtGetNextStruct ( );

SOMTTypedefEntryC somtGetNextTypedef ( );

SOMTUnionEntryC somtGetNextUnion ( );

SOMTEntryC somtGetNextPubdef ( );

Note: These methods do *not* take an **Environment** parameter.

Description

The **somtGetNext**<*Item*> methods return the next item (of the type shown above) for the entry represented by *receiver*, if it has a next item of that type. Otherwise, it returns NULL. The **somtGetNextPubdef** method returns the next constant/type definition of the class, whether a typedef, struct, union, etc.

A call to a **somtGetNext**<*Item*> method is relative to the last call of either the same method or the corresponding **somtGetFirst**<*Item*> method**,** applied to the same entry object. Note that this implies that the same **somtGetFirst**<*Item*> and **somtGetNext**<*Item*> methods cannot be used in doubly nested loops. For example, the following doubly nested loop will not work, because following the first execution of the inner loop, the invocation of **somtGetNextMethod** in the outer loop will return NULL:

```
for (m1 = _somtGetFirstMethod(cls); m1;
     m1 = _somtGetNextMethod(cls))
   for (m2 = _somtGetFirstMethod(cls); m2;
        m2 = _somtGetNextMethod(cls))
    /* etc. */
```

Nested loops such as the one above are permissible if the target object (e.g., “cls”) of the inner loop differs from the target object of the outer loop, or if a different **somtGetFirst**<*Item*> method is used in the inner loop (for instance, a **somtGetNextParameter** loop can be nested inside a **somtGetNextMethod** loop).

Parameters

receiver The entry whose next item is to be retrieved.

Return Value

These methods return the next item (of the type shown above) for the entry represented by *receiver*, if it has a next item of that type. Otherwise, it returns NULL. The type of item returned is specific to the method; see the method call syntax shown above

Example

To iterate through the base classes:

```
SOMTBaseClassEntryC myEntry;

printf("List of base classes:\n");
for (myEntry = _somtGetFirstBaseClass(cls); myEntry;
    myEntry = _somtGetNextBaseClass(cls))
    printf("%s\n", __get_somtEntryName(myEntry));
```

Related Information

Methods: somtGetFirst<*Item*>

somtGetReleaseNameList Method

Purpose

Gets the release-order list of a class.

IDL Syntax

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

Note: This method does *not* take an **Environment** parameter.

Description

The **somtGetReleaseNameList** method puts the release-order list of the specified class in *buffer*. Names in the list are delimited by newlines so that the list can be used as a symbol value suitable for list substitution. Users must allocate enough space for the *buffer*; no tests are made to assure that adequate space has been allocated. Upon completion, **somtGetReleaseNameList** returns the number of release-order names stored in *buffer*.

Parameters

receiver	An object of class **SOMTClassEntryC** representing a class.
buffer	The address of a character buffer in which to store the release-order list.

Return Value

The **somtGetReleaseNameList** method returns the number of names stored in *buffer*.

Original Class

SOMTClassEntryC

Related Information

Methods: somtGetFirstReleaseName, somtGetNextReleaseName

SOMTCommonEntryC Class

Description

The **SOMTCommonEntryC** class defines methods and attributes that are common to its subclasses — **SOMTMethodEntryC**, **SOMTDataEntryC**, **SOMTUserDefinedTypeEntryC,** and **SOMTParameterEntryC**. Entry objects that an emitter uses are actually instances of one of these subclasses, rather than of **SOMTCommonEntryC** itself. The **SOMTCommonEntryC** class provides attributes and methods for accessing type information.

File Stem

sccommon

Base

SOMTEntryC

Metaclass

SOMClass

Ancestor Classes

SOMTEntryC, **SOMObject**

Attributes

Listed below is each available attribute, with its corresponding type in parentheses, followed by a description of its purpose.

somtTypeObj (SOMTEntryC)

A pointer to the object representing the type of the entry. This attribute may be NULL when processing an input file containing an OIDL, rather than an IDL, interface specification. This attribute has no “Set” method.

somtType (string)

The IDL type of the entry**,** in string form. For methods, this is the return type; for data, parameters, or user-defined types, it is the type. It is in the form

<typename><pointer stars> <array– declarators>.

This attribute has no “Set” method.

somtPtrs (string)

The string of stars associated with a pointer type. For example, a type “short *” has somtPtrs = “*”, a type “short **” has somtPtrs = “**”, etc.
If the type of the entry is not a pointer, then somtPtrs = NULL.
This attribute may be NULL when processing an input file containing an OIDL, rather than an IDL, interface specification. This attribute has no “Set” method.

somtArrayDimsString (string)

The array dimensions, as a string, for entries of array type.
This attribute has no “Set” method.

New Methods

somtIsArray
somtIsPointer
somtGetFirstArrayDimension
somtGetNextArrayDimension

Overriding Methods

somtSetSymbolsOnEntry
somDumpSelfInt

somtGetFirstArrayDimension Method

Purpose

The **somtGetFirstArrayDimension** method gets the first array dimension for a particular entry.

IDL Syntax

unsigned long somtGetFirstArrayDimension ( );

Note: This method does *not* take an **Environment** parameter.

Description

The **somtGetFirstArrayDimension** method returns the first array dimension for the entry on which the method is invoked, if it has one. Otherwise, it returns zero. The next array dimension can be obtained using the corresponding **somtGetNextArrayDimension** method.

Note that the **somtGetFirstArrayDimension** and **somtGetNextArrayDimension** methods cannot be used in doubly nested loops. For example, the following doubly nested loop will not work, because following the first execution of the inner loop, the invocation of **somtGetNextArrayDimension** in the outer loop will return zero:

```
for (ad1 = _somtGetFirstArrayDimension(entry); ad1;
     ad1 = _somtGetNextArrayDimension(entry))
   for (ad2 = _somtGetFirstArrayDimension(entry); ad2;
        ad2 = _somtGetNextArrayDimension(entry))
    /* etc. */
```

Nested loops such as the one above are permissible if the target object (e.g., “entry”) of the inner loop differs from the target object of the outer loop, or if a different **somtGetFirst**<*Item*> method is used in the inner loop (for instance, a **somtGetNextParameter** loop can be nested inside a **somtGetNextArrayDimension** loop).

The **somtGetFirstArrayDimension** method may not be reliable when processing an OIDL, rather than an IDL, interface specification.

Parameters

receiver The entry whose first array dimension is to be retrieved.

Return Value

The **somtGetFirstArrayDimension** method returns the first array dimension for a particular entry (method, parameter, user-defined type, etc.), if it has one; otherwise, it returns zero.

Example

To iterate through the array dimensions of a method:

```
unsigned long n;

printf(”List of array dimensions:\n”);
for (n = _somtGetFirstArrayDimension(method); n;
    n = _somtGetNextArrayDimension(method))
    printf(”[%lu]”, n);
```

Related Information

Methods: somtGetNextArrayDimension

somtGetNextArrayDimension Method

Purpose

Gets the next array dimension for a particular entry, relative to the previous call for a similar entry.

IDL Syntax

unsigned long somtGetNextArrayDimension ( );

Note: This method does *not* take an **Environment** parameter.

Description

The **somtGetNextArrayDimension** method returns the next array dimension for the entry represented by *receiver*, if it has a next dimension. Otherwise, it returns zero.

A call to a **somtGetNextArrayDimension** method is relative to the last call of either the same method or the corresponding **somtGetFirstArrayDimension** method**,** applied to the same entry object. Note that this implies that the same **somtGetFirstArrayDimension** and **somtGetNextArrayDimension** methods cannot be used in doubly nested loops. For example, the following doubly nested loop will not work, because following the first execution of the inner loop, the invocation of **somtGetNextArrayDimension** in the outer loop will return zero:

```
for (ad1 = _somtGetFirstArrayDimension(entry); ad1;
     ad1 = _somtGetNextArrayDimension(entry))
   for (ad2 = _somtGetFirstArrayDimension(entry); ad2;
        ad2 = _somtGetNextArrayDimension(entry))
    /* etc. */
```

Nested loops such as the one above are permissible if the target object (e.g., “entry”) of the inner loop differs from the target object of the outer loop, or if a different **somtGetFirst**<*Item*> method is used in the inner loop (for instance, a **somtGetNextParameter** loop can be nested inside a **somtGetNextArrayDimension** loop).

Parameters

receiver The entry whose next array dimension is to be retrieved.

Return Value

The **somtGetNextArrayDimension** method returns the next array dimension for the specified entry (method, parameter, user-defined type, etc.), if it has a next dimension. Otherwise, it returns zero.

Example

To iterate through the array dimensions of a method:

```
unsigned long n;

printf(”List of array dimensions:\n”);
for (n = _somtGetFirstArrayDimension(method); n;
    n = _somtGetNextArrayDimension(method))
    printf(”[%lu]”, n);
```

Related Information

Methods: somtGetFirstArrayDimension

somtIsArray Method

Purpose

Tests to determine whether the type of a method, data item, user-defined type, attribute declarator, struct member declarator, or parameter is an array. If so, its size is returned.

IDL Syntax

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

Note: This method does *not* take an **Environment** parameter.

Description

The **somtIsArray** method has a dual purpose. If the type of the *receiver* involves an array, then **somtIsArray** returns TRUE and sets *size* to the size (in the first dimension) of the array encountered. If no array is encountered, then **somtIsArray** returns FALSE.

Parameters

receiver — An object of class **SOMTCommonEntryC** representing a method, data item, user-defined type, declarator, or parameter to test.

size — The address where the size of the array should be stored.

Return Value

If the specified entry's type is an array, then the **somtIsArray** method returns TRUE and *size* is set to the size (in the first dimension) of the array. Otherwise, **somtIsArray** returns FALSE.

Original Class

SOMTCommonEntryC

Related Information

Methods: somtIsPointer

somtIsPointer Method

Purpose

Tests whether the type of a method, data item, user-defined type, attribute declarator, struct member declarator, or parameter is a pointer.

IDL Syntax

boolean somtIsPointer ( );

Note: This method does *not* take an **Environment** parameter.

Description

The **somtIsPointer** method returns TRUE if the type of the **SOMTCommonEntryC** object is a pointer. Otherwise, it returns FALSE.

Parameters

receiver An object of class **SOMTCommonEntryC** representing the entry to be tested.

Return Value

The **somtIsPointer** method returns TRUE if the type of the **SOMTCommonEntryC** object is a pointer. Otherwise, it returns FALSE.

Example

```
SOMTCommonEntryC myEntry;

if(somtIsPointer(myEntry)
   printf("Saw a pointer.\n");
else
  printf("Didn't see a pointer!\n");
```

Original Class

SOMTCommonEntryC

Related Information

Methods: somtIsArray

SOMTConstEntryC Class

Description

A **SOMTConstEntryC** object represents a constant definition. It provides attributes for accessing the type and value of the constant.

File Stem

scconst

Base

SOMTEntryC

Metaclass

SOMClass

Ancestor Classes

SOMTEntryC, **SOMObject**

Attributes

Listed below is each available attribute, with its corresponding type in parentheses, followed by a description of its purpose:

somtConstTypeObj (SOMTEntryC)
A pointer to an object representing the type of the constant's value. This attribute has no "set" method.

somtConstType (string)
The type of the constant's value, as a string. This attribute has no "set" method.

somtConstStringVal (string)
The (unevaluated) value of the constant, as a string. (Constants within the value expression are replaced with their values, however.) The value of constants of type string or char do not include the quotes, unlike the **somtConstVal** attribute. This attribute has no "set" method.

somtConstVal (string)
The (evaluated) value of the constant, as a string. (For constants of type string or char, the **somtConstVal** attribute includes the quotes, while the **somtConstStringVal** attribute does not.) This attribute has no "set" method. The "get" method returns a string whose ownership is transferred to the caller.

somtConstIsNegative (boolean)
Whether the constant's value is a negative (short or long) integer.

somtConstNumVal (unsigned long)
The numeric value of the constant. This attribute has no "set" method. This attribute should only be used if the value of the constant can be represented by an unsigned long (i.e., not if its type is "string," "float," or "double," or if its **somtConstIsNegative** attribute is TRUE).

somtConstNumNegVal (long)
The numeric value of the constant, if it is a negative (short or long) integer. This attribute has no "set" method. This attribute should be used instead of **somtConstNumVal** if the value of the constant is negative (i.e., if its **somtConstIsNegative** attribute is TRUE).

New Methods

None.

Overriding Methods

somtSetSymbolsOnEntry
somDumpSelfInt

SOMTDataEntryC Class

Description

A **SOMTDataEntryC** object represents either an internal instance data declaration in the "implementation" section of a SOM IDL class interface definition or a declarator of an attribute or struct member.

File Stem

scdata

Base

SOMTCommonEntryC

Metaclass

SOMClass

Ancestor Classes

SOMTCommonEntryC, **SOMTEntryC**, **SOMObject**

Attributes

Listed below is each available attribute, with its corresponding type in parentheses, followed by a description of its purpose. None of the following attributes has a "set" method.

somtIsSelfRef (boolean)

Whether a declarator of a structure member is self-referential (pointing to the same type of structure for which it is a member).
This attribute has no "Set" method.

New Methods

None.

Overriding Methods

somtSetSymbolsOnEntry
somDumpSelfInt

SOMTEmitC Class

Description

An object of the **SOMTEmitC** class represents an emitter. A new type of emitter can be constructed by subclassing this class and overriding the **somtGenerateSections** method (and other methods, if necessary).

An emitter has as attributes (see below): a *target file*, a *target class* or *target module*, and a *template*. The target file is the file to which output will be directed. The target class/module (the class/module about which information will be emitted) is represented by an object of class **SOMTClassEntryC** or **SOMTModuleEntryC** (depending on whether the emitter is being invoked for a class or a module). This object is constructed by the SOM Compiler when it compiles the IDL specification. The **SOMTClassEntryC** and **SOMTModuleEntryC** classes provide methods for accessing information found in the IDL specification of the target class/module.

The template of the emitter defines the format and content of the sections that the emitter produces. (The emitter itself controls which sections are actually emitted and their order, by its implementation of the **somtGenerateSections** method.) The template is represented by an object of class **SOMTTemplateOutputC** that is initialized from the template file (typically a .efw file). The template is defined in terms of symbols that, when emitted, are replaced by values appropriate for the emitter's target class/module. The **SOMTemplateOutputC** class defines several general-purpose symbols as well as methods through which an emitter can define special-purpose symbols. An emitter's template also maintains the emitter's symbol table.

The **SOMTEmitC** class provides methods for emitting sections that are common to many emitter templates (sections for attribute definitions, method definitions, etc.). These methods are listed in the *sections* group, below. A subclass of **SOMTEmitC** can override these methods to change the way that a particular section is emitted. The **SOMTTemplateOutputC** class provides methods for defining and emitting special-purpose sections.

Emitter Sections

The **SOMTEmitC** class provides methods for emitting the following template sections. The default section name is given in parentheses. By convention, section names end in **S**.

Prolog Describes text to be emitted before any other sections (**prologS**).

Base Includes Determines how base (parent) class #include statements are emitted (**baseIncludesS**).

Meta Include Determines how a metaclass #include statement is emitted (**metaIncludeS**).

Class Determines what information about the class as a whole is emitted (**classS**).

Base Determines what information about a base (parent) classes of a class is emitted (**baseS**).

Meta Determines what information about the class's metaclass is emitted (**metaS**).

Constant Determines what information about user-defined constants is emitted (**constantS**).

Typedef Determines what information about user-defined types is emitted (**typedefS**).

Struct Determines what information about user-defined structs is emitted (**structS**).

Union Determines what information about user-defined unions is emitted (**unionS**).

Enum Determines what information about user-defined enumerations is emitted (**enumS**).

Attribute Determines what information about the class's attributes is emitted (**attributeS**).

Methods Determines what information about the methods of a class is emitted (**methodsS**). More specialized method sections can be specified using **inheritedMethodsS** or **overrideMethodsS**).

Release Determines how information about the release order statement of a class definition is emitted (**releaseS**).

Passthru Determines what information about passthru statements is emitted (**passthruS**).

Data Determines what information about internal instance variables of a class is emitted (**dataS**).

Interface Determines what information about the interfaces in a module is emitted (**interfaceS**).

Module Determines what information about a module is emitted (**moduleS**).

Epilog Describes text to be emitted after all other sections are emitted (**epilogS**).

Repeating Sections

Some sections apply to a variable number of items that must be dealt with iteratively. This can be true of the *base* section (since a class can have more that one base class), as well as the *base includes, constant, typedef, struct, union, enum, methods, data*, *passthru*, *interface* and *module* sections. These repeating sections can be preceded by a *prolog* (information to be emitted *prior* to iterating over the items), and followed by an *epilog* (information to be emitted *after* iterating over the items). The **SOMTEmitC** class provides methods for emitting the following prolog and epilog sections:

basePrologS, baseEpilogS, baseIncludesPrologS, baseIncludesEpilogS, constantPrologS, constantEpilogS, typedefPrologS, typedefEpilogS, structPrologS, structEpilogS, unionPrologs, unionEpilogS, enumPrologS, enumEpilogS, passthruPrologS, passthruEpilogS, dataPrologS, dataEpilogS, attributePrologS, attributeEpilogS, methodsPrologS, methodsEpilogS, interfacePrologS, interfaceEpilogS, modulePrologS, moduleEpilogS.

An emitter typically emits a repeating section as follows:

1. The *prolog* section, if any, is emitted.
2. The emitter iterates over each item to be described, emitting the section body for each.
3. The *epilog* section, if any, is emitted.

To facilitate emitting repeating sections, the **SOMTEmitC** class provides a set of *scanning methods* (listed below under "Scanning") that perform the above steps for a particular repeating section.

File Stem

scemit

Base

SOMObject

Metaclass

SOMClass

Ancestor Classes

SOMObject

Attributes

Listed below is each available attribute, with its corresponding type in parentheses, followed by a description of its purpose:

somtTemplate (SOMTTemplateOutputC)
The template for the emitter.

somtTargetFile (FILE)
The target file for the emitter.

somtTargetClass (SOMTClassEntryC)
The target class for the emitter.
If the emitter is handling a module rather than a class, then this attribute is NULL.

somtTargetModule (SOMTModuleEntryC)
The target module for the emitter.
If the emitter is handling a class rather than a module, then this attribute is NULL.

somtEmitterName (string)
The name by which the emitter is invoked (using the "–s" option of the "sc" command). This is typically the filestem of the emitter's .idl file.

New Methods

Group: framework

somtGenerateSections
somtOpenSymbolsFile

somtSetPredefinedSymbols
somtFileSymbols
somtGetGlobalModifierValue
somtGetFirstGlobalDefinition
somtGetNextGlobalDefinition

Group: sections

somtEmitProlog
somtEmitBaseIncludesProlog
somtEmitBaseIncludes
somtEmitBaseIncludesEpilog
somtEmitMetaInclude
somtEmitClass
somtEmitMeta
somtEmitBaseProlog
somtEmitBase
somtEmitBaseEpilog
somtEmitConstantProlog
somtEmitConstant
somtEmitConstantEpilog
somtEmitTypedefProlog
somtEmitTypedef
somtEmitTypedefEpilog
somtEmitStructProlog
somtEmitStruct
somtEmitStructEpilog
somtEmitUnionProlog
somtEmitUnion
somtEmitUnionEpilog
somtEmitEnumProlog
somtEmitEnum

somtEmitEnumEpilog
somtEmitAttributeProlog
somtEmitAttribute
somtEmitAttributeEpilog
somtEmitFullPassthru
somtEmitPassthruProlog
somtEmitPassthru
somtEmitPassthruEpilog
somtEmitRelease
somtEmitDataProlog
somtEmitData
somtEmitDataEpilog
somtEmitMethodsProlog
somtEmitMethod
somtEmitMethods
somtEmitMethodsEpilog
somtEmitInterfaceProlog
somtEmitInterface
somtEmitInterfaceEpilog
somtEmitModuleProlog
somtEmitModule
somtEmitModuleEpilog
somtEmitEpilog

Group: scanning

somtScanBases
somtScanBasesF
somtScanConstants
somtScanTypedefs
somtScanStructs
somtScanUnions
somtScanEnums
somtScanAttributes
somtScanMethods
somtScanData
somtScanDataF
somtScanPassthru
somtScanInterfaces
somtScanModules

Group: filters

somtNew
somtImplemented
somtOverridden
somtInherited
somtAll
somtNewProc
somtNewNoProc
somtVA

Overriding Methods

somInit,
somUninit,
somDumpSelfInt

somtAll Method

Purpose

Checks whether the target class of an emitter supports a specified instance method.

IDL Syntax

boolean somtAll (
in SOMTMethodEntryC *method***);**

Note: This method does *not* take an **Environment** parameter.

Description

The **somtAll** method checks whether the target class of an emitter supports a specified instance method (whether the method can be invoked on instances of the class).

Parameters

receiver An object of class **SOMTEmitC** representing an emitter.

method An object of class **SOMTMethodEntryC** representing a method to be tested.

Return Value

The **somtAll** method returns TRUE if the emitter's target class supports the specified method. Otherwise, it returns FALSE.

Example

```
_somtScanMethods(emitter, "somtAll", "somtEmitMethodsProlog",
                 "somtEmitMethod", "somtEmitMethodsEpilog",TRUE);
```

Original Class

SOMTEmitC

Related Information

Methods: somtOverridden, somtInherited, somtImplemented, somtNew, somtNewProc, somtNewNoProc, somtVA, somtScanMethods

somtEmit<*Section*> Methods

Purpose

These methods emit a particular section from an emitter's template.

IDL Syntax

void somtEmitAttribute (in SOMTAttributeEntryC *entry***);**
void somtEmitAttributeEpilog ( );
void somtEmitAttributeProlog ( );
void somtEmitBase (in SOMTBaseClassEntryC *entry***);**
void somtEmitBaseEpilog ( );
void somtEmitBaseIncludes (in SOMTBaseClassEntryC *entry***);**
void somtEmitBaseIncludesEpilog ( );
void somtEmitBaseIncludesProlog ( );
void somtEmitBaseProlog ( );
void somtEmitClass ( );
void somtEmitConstant (in SOMTConstEntryC *entry***);**
void somtEmitConstantEpilog ( );
void somtEmitConstantProlog ( );
void somtEmitData (in SOMTDataEntryC *entry***);**
void somtEmitDataEpilog ( );
void somtEmitDataProlog ( );
void somtEmitEnum (in SOMTEnumEntryC *entry***);**
void somtEmitEnumEpilog ( );
void somtEmitEnumProlog ( );
void somtEmitEpilog ( );
void somtEmitInterface (in SOMTClassEntryC *entry***);**
void somtEmitInterfaceEpilog ( );
void somtEmitInterfaceProlog ( );
void somtEmitMeta ( );
void somtEmitMetaInclude ( );
void somtEmitMethod (in SOMTMethodEntryC *entry***);**
void somtEmitMethods (in SOMTMethodEntryC *entry***);**
void somtEmitMethodsEpilog ( );
void somtEmitMethodsProlog ( );
void somtEmitModule (in SOMTModuleEntryC *entry***);**
void somtEmitModuleEpilog ( );
void somtEmitModuleProlog ( );
void somtEmitPassthru (in SOMTPassthruEntryC *entry***);**
void somtEmitPassthruEpilog ( );

void somtEmitPassthruProlog ( );

void somtEmitProlog ( );

void somtEmitRelease ( );

void somtEmitStruct (in SOMTStructEntryC *entry***);**

void somtEmitStructEpilog ( );

void somtEmitStructProlog ( );

void somtEmitTypedef (in SOMTTypedefEntryC *entry***);**

void somtEmitTypedefEpilog ( );

void somtEmitTypedefProlog ( );

void somtEmitUnion (in SOMTUnionEntryC *entry***);**

void somtEmitUnionEpilog ( );

void somtEmitUnionProlog ( );

Note: These methods do *not* take an **Environment** parameter.

Description

The **somtEmit**<*section*> methods emit a particular section from an emitter's template, using the **somtOutputSection** method. In the case of a repeating section, an entry object is passed as a parameter; prior to emitting the section, the **somtSetSymbolsOnEntry** method is invoked on that entry so that the template symbols used in the section to be emitted will correspond to that entry.

The section emitted by each of these methods is determined by a section-name symbol, as shown in the adjacent table. To change the section to be emitted by a particular method, change the value of the corresponding section-name symbol (using **somtSetSymbolCopyValue**) prior to invoking the section-emitting method. For example, to have the **somtEmitBase** method emit the "mybaseS" section of the template instead of the "baseS" section, which is the default, set the "baseSN" symbol to "mybaseS" prior to invoking **somtEmitBase**.

The **somtEmitMethod** method emits a specialized methods section for the specified method, according to whether the method is new, inherited, or overriding. (The specialized sections are, by default, **methodsS**, **inheritedMethodsS**, and **overrideMethodsS**, depending on the value of section-name symbols "methodsSN," "inheritedMethodsSN," and "overrideMethodsSN.") The **somtEmitMethods** method, by contrast, emits the generic methods section for the specified method. The generic methods section is determined by the value of the "methodsSN" symbol, which is by default **methodsS**.

Parameters

receiver	An object of class **SOMTEmitC** representing an emitter.
entry	An entry object to be used to set the values of the symbols used in the template section to be emitted.

Return Value

None.

Original Class

SOMTEmitC

Related Information

Methods: somtEmitFullPassthru, somtScan<*Section*>

Symbol Name	Initial Value (Section Name)	Used by Method
attributeSN	attributeS	somtEmitAttribute
attributeEpilogSN	attributeEpilogS	somtEmitAttributeEpilog
attributePrologSN	attributePrologS	somtEmitAttributeProlog
baseSN	baseS	somtEmitBase
baseEpilogSN	baseEpilogS	somtEmitBaseEpilog
baseIncludesSN	baseIncludesS	somtEmitBaseIncludes
baseIncludesEpilogSN	baseIncludesEpilogS	somtEmitBaseIncludesEpilog
baseIncludesPrologSN	baseIncludesPrologS	somtEmitBaseIncludesProlog
basePrologSN	basePrologS	somtEmitBaseProlog
classSN	classS	somtEmitClass
constantSN	constantS	somtEmitConstant
constantPrologSN	constantPrologS	somtEmitConstantProlog
constantEpilogSN	constantEpilogS	somtEmitConstantEpilog
dataSN	dataS	somtEmitData
dataEpilogSN	dataEpilogS	somtEmitDataEpilog
dataPrologSN	dataPrologS	somtEmitDataProlog
enumSN	enumS	somtEmitEnum
enumEpilogSN	enumEpilogS	somtEmitEnumEpilog
enumPrologSN	enumPrologS	somtEmitEnumProlog
epilogSN	epilogS	somtEmitEpilog
inheritedMethodsSN	inheritedMethodsS	somtEmitMethod
interfaceSN	interfaceS	somtEmitInterface
interfaceEpilogSN	interfaceEpilogS	somtEmitInterfaceEpilog
interfacePrologSN	interfacePrologS	somtEmitInterfaceProlog
metaSN	metaS	somtEmitMeta
metaIncludeSN	metaIncludeS	somtEmitMetaIncludes
methodsSN	methodsS	somtEmitMethod
methodsSN	methodsS	somtEmitMethods
methodsEpilogSN	methodsEpilogS	somtEmitMethodsEpilog
methodsPrologSN	methodsPrologS	somtEmitMethodsProlog
moduleSN	moduleS	somtEmitModule
moduleEpilogSN	moduleEpilogS	somtEmitModuleEpilog
modulePrologSN	modulePrologS	somtEmitModuleProlog
overrideMethodsSN	overrideMethodsS	somtEmitMethod
passthruSN	passthruS	somtEmitPassthru
passthruEpilogSN	passthruEpilogS	somtEmitPassthruEpilog
passthruPrologSN	passthruPrologS	somtEmitPassthruProlog
prologSN	prologS	somtEmitProlog
releaseSN	releaseS	somtEmitRelease
structSN	structS	somtEmitStruct
structEpilogSN	structEpilogS	somtEmitStructEpilog
structPrologSN	structPrologS	somtEmitStructProlog
typedefSN	typedefS	somtEmitTypedef
typedefEpilogSN	typedefEpilogS	somtEmitTypedefEpilog
typedefPrologSN	typedefPrologS	somtEmitTypedefProlog
unionEpilogSN	unionEpilogS	somtEmitUnionEpilog
unionSN	unionS	somtEmitUnion
unionPrologsN	unionPrologs	somtEmitUnionProlog

somtEmitFullPassthru Method

Purpose

Emits the passthru section for each of a class's passthrus having a particular target and before/after classification.

IDL Syntax

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

Note: This method does *not* take an **Environment** parameter.

Description

The **somtEmitFullPassthru** method emits the passthru section for each passthru item defined for an emitter's target class that has the specified target language and the specified before/after setting and whose target is the emitter on which the method is invoked.

The **somtIsBeforePassthru** method is used to check that passthru entries match the *before* parameter. The **somtPassthruLanguage** attribute of each passthru entry is matched against the *language* parameter. The **somtPassthruTarget** attribute of each passthru entry is matched against the **somtEmitterName** attribute of the emitter on which the method is invoked.

This method first invokes the **somtEmitPassthruProlog** method. Then, for each passthru entry that satisfies the above requirements, this method invokes the **somtEmitPassthru** method. Finally, it invokes the **somtEmitPassthruEpilog** method.

Parameters

receiver — An object of class **SOMTEmitC** representing an emitter.

before — Whether before (TRUE) or after (FALSE) passthrus are to be emitted.

language — The target language of the passthrus for which sections are to be emitted (in upper case only). "C" is used for both C and C++.

Return Value

None.

Example

```
__set_somtEmitterName(emitter, "mine");
__set_somtTargetClass(emitter, oCls);
__set_somtTargetFile(emitter, fp);
_somtEmitFullPassthru(emitter, TRUE, "C");
```

Original Class

SOMTEmitC

Related Information

Methods: somtEmitPassthruProlog, **somtEmitPassthruEpilog, somtEmitPassthru, somtIsBeforePassthru**

somtFileSymbols Method

Purpose

Sets predefined symbols that have a single value in all sections of the output template.

IDL Syntax

void somtFileSymbols ( );

Description

The **somtFileSymbols** method sets predefined symbols that have a single value in all sections of the output template. This includes symbols for the target class or target module, symbols for the metaclass of the target class, if any, and special symbols like <timeStamp>.

Parameters

receiver An object of class **SOMTEmitC** representing an emitter.

Return Value

None.

Example

```
emitter = MyEmitterNew();
__set_somtTargetFile(emitter, fp);
__set_somtTargetClass(emitter, oCls);
_somtFileSymbols(emitter);
```

Original Class

SOMTEmitC

somtGenerateSections Method

Purpose

Calls each of the section-emitting methods.

IDL Syntax

boolean somtGenerateSections ( );

Note: This method does *not* take an **Environment** parameter.

Description

The default implementation of **somtGenerateSections** method calls each of the section-emitting methods in the following order:

For emitters having a target class:

somtEmitProlog
somtEmitBaseIncludesProlog
somtEmitBaseIncludes — For each base class.
somtEmitBaseIncludesEpilog
somtEmitMetaInclude
somtEmitClass
somtEmitBaseProlog
somtEmitBase — For each base class.
somtEmitBaseEpilog
somtEmitMeta
somtEmitConstantProlog
somtEmitConstant — For each constant.
somtEmitConstantEpilog
somtEmitTypedefProlog
somtEmitTypedef — For each typedef.
somtEmitTypedefEpilog
somtEmitStructProlog
somtEmitStruct — For each struct.
somtEmitStructEpilog
somtEmitUnionProlog
somtEmitUnion — For each union.
somtEmitUnionEpilog
somtEmitEnumProlog
somtEmitEnum — For each enum.
somtEmitEnumEpilog
somtEmitAttributeProlog
somtEmitAttribute — For each attribute.
somtEmitAttributeEpilog
somtEmitMethodsProlog
somtEmitMethod — For each method.
somtEmitMethodsEpilog
somtEmitRelease
somtEmitPassthruProlog
somtEmitPassthru — For each passthru item.
somtEmitPassthruEpilog
somtEmitDataProlog
somtEmitData — For each data item.
somtEmitDataEpilog
somtEmitEpilog

For emitters that have a target module:

somtEmitProlog
somtEmitConstantProlog
somtEmitConstant — For each constant.
somtEmitConstantEpilog
somtEmitTypedefProlog
somtEmitTypedef — For each typedef.
somtEmitTypedefEpilog
somtEmitStructProlog
somtEmitStruct — For each struct.
somtEmitStructEpilog
somtEmitUnionProlog
somtEmitUnion — For each union.
somtEmitUnionEpilog
somtEmitEnumProlog
somtEmitEnum — For each enum.
somtEmitEnumEpilog
somtEmitInterfaceProlog
somtEmitInterface — For each interface.
somtEmitInterfaceEpilog
somtEmitModuleProlog
somtEmitModule — For each embedded module.
somtEmitModuleEpilog
somtEmitEpilog

Repeating sections (such as **somtEmitBase**) are emitted using the corresponding **somtScan**<*Section*> method.

To rearrange the order of sections, or to add or delete sections in your emitter, override the **somtGenerateSections** method.

Parameters

receiver An object of class **SOMTEmitC** representing an emitter.

Return Value

The **somtGenerateSections** method returns TRUE.

Original Class

SOMTEmitC

Related Information

Methods: somtEmit<*Section*>**, somtEmitFullPassthru, somtScan**<*Section*>

somtGetFirstGlobalDefinition Method

Purpose

Returns the first **type, constant, exception,** or **forward** declaration that is *not* associated with any interface or module.

IDL Syntax

SOMTEntryC somtGetFirstGlobalDefinition ( );

Note: This method does *not* take an **Environment** parameter.

Description

The **somtGetFirstGlobalDefinition** method returns the first **type, constant, exception,** or **forward** declaration that is *not* associated with any interface or module. Note that **type** declarations also include **struct** and **union** declarations. In the IDL source file, these global declarations must be surrounded by **#pragma somemittypes** statements for them to be visible via this method. For example:

```
#pragma somemittypes on
    typedef sequence <long,10> vec10;
    exception BAD_FLAG { long ErrCode; char Reason[80]; };
    typedef long long_t;
#pragma somemittypes off
```

Parameters

receiver A pointer to an object of class **SOMTEmitC** representing an emitter.

Return Value

The **somtGetFirstGlobalDefinition** method returns a pointer to an object of class **SOMTEntryC** representing the first global **type, constant, exception,** or **forward** declaration in the receiver.

Example

```
SOMTEntryC  entry;

for (entry = _somtGetFirstGlobalDefinition(emitter);
     entry;  entry = _somtGetNextGlobalDefinition(emitter))
    if ((__get_somtElementType(entry) == SOMTClassE) &&
         __get_somtForwardRef(entry))
            /* "entry" represents a forward declaration for
             *  a class; handle it appropriately
             */
    else
        /* "entry" represents something else, such as a
         * constant, typedef, etc.  Determine what it
         * represents using __get_somtElementType as above
         * and handle it appropriately.
         */
```

Original Class

SOMTEmitC

Related Information

Methods: somtGetNextGlobalDefinition

somtGetGlobalModifierValue Method

Purpose

Gets the value of a global modifier specified via the **–m** option when the SOM Compiler is invoked.

IDL Syntax

string somtGetGlobalModifierValue (in string *modifierName***);**

Note: This method does *not* take an **Environment** parameter.

Description

The **somtGetGlobalModifierValue** method gets the value of a global modifier specified via the **–m** option when the SOM Compiler is invoked. For example,

```
sc –m"foo=bar"  file.idl
```

specifies to the SOM Compiler 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 with no value (for modifiers that are boolean), as in

```
sc –mfoo file.idl
```

then the result of this method will be non-NULL. If the requested modifier was not specified in the **sc** command, then the result of this method is NULL.

Parameters

receiver An object of class **SOMTEmitC** representing an emitter.

modifierName The name of the global modifier whose value is needed.

Return Value

The value of the global modifier is returned. If the modifier is present but has no value (for modifiers that are boolean), the result of this method will be non-NULL. If the requested modifier was not specified in the *sc* command, then the result of this method is NULL.

Original Class

SOMTEmitC

somtGetNextGlobalDefinition Method

Purpose

Returns the next **type, constant, exception,** or **forward** declaration that is *not* associated with any interface or module, relative to a similar, preceding call.

IDL Syntax

SOMTEntryC somtGetNextGlobalDefinition ( );

Note: This method does *not* take an **Environment** parameter.

Description

The **somtGetNextGlobalDefinition** method returns the next **type, constant, exception,** or **forward** declaration that is *not* associated with any interface or module, relative to a prior call to method **somtGetFirstGlobalDefinition** or **somtGetNextGlobalDefinition**. In the IDL source file, these global declarations must be surrounded by **#pragma somemittypes** statements for them to be visible via this method.

Parameters

receiver A pointer to an object of class **SOMTEmitC** representing an emitter.

Return Value

The **somtGetNextGlobalDefinition** method returns a pointer to an object of class **SOMTEntryC** representing the next global **type, constant, exception,** or **forward** declaration in the receiver, relative to a previous call to either method **somtGetFirstGlobalDefinition** or **somtGetNextGlobalDefinition**.

Example

```
SOMTEntryC  entry;

for (entry = _somtGetFirstGlobalDefinition(emitter);
     entry;  entry = _somtGetNextGlobalDefinition(emitter))
    if ((__get_somtElementType(entry) == SOMTClassE) &&
         __get_somtForwardRef(entry))
            /* "entry" represents a forward declaration for
             *  a class; handle it appropriately
             */
    else
        /* "entry" represents something else, such as a
         * constant, typedef, etc.  Determine what it
         * represents using __get_somtElementType as above
         * and handle it appropriately.
         */
```

Original Class

SOMTEmitC

Related Information

Methods: somtGetFirstGlobalDefinition

somtImplemented Method

Purpose

Checks whether the target class of an emitter introduces or overrides a specified method.

IDL Syntax

boolean somtImplemented (
in SOMTMethodEntryC *method***);**

Note: This method does *not* take an **Environment** parameter.

Description

The **somtImplemented** method checks whether the target class of the receiver introduces or overrides the method specified by *method*.

Parameters

receiver An object of class **SOMTEmitC** representing an emitter.

method An object of class **SOMTMethodEntryC** representing the method to be tested.

Return Value

The **somtImplemented** method returns TRUE if the emitter's target class introduces or overrides the specified method. Otherwise, it returns FALSE.

Example

```
_somtScanMethods(emitter, "somtImplemented",
                 "somtEmitMethodsProlog", "somtEmitMethod",
                 "somtEmitMethodsEpilog",TRUE);
```

Original Class

SOMTEmitC

Related Information

Methods: somtAll, somtInherited, somtOverridden, somtNew, somtNewNoProc, somtNewProc, somtVA, somtScanMethods

somtInherited Method

Purpose

Checks whether the target class of an emitter inherits a specified method.

IDL Syntax

boolean somtInherited (
in SOMTMethodEntryC *method***);**

Note: This method does *not* take an **Environment** parameter.

Description

The **somtInherited** method checks whether the target class of an emitter inherits a specified method.

Parameters

receiver An object of class **SOMTEmitC** representing an emitter.

method An object of class **SOMTMethodEntryC** representing the method to be tested.

Return Value

The **somtInherited** method returns TRUE if the emitter's target class inherits the specified method. Otherwise, it returns FALSE.

Example

```
_somtScanMethods(emitter, "somtInherited",
                  "somtEmitMethodsProlog",
                 "somtEmitMethod",
                 "somtEmitMethodsEpilog",TRUE);
```

Original Class

SOMTEmitC

Related Information

Methods: somtAll, somtOverridden, somtImplemented, somtNew, somtNewNoProc, somtNewProc, somtVA, somtScanMethods

somtNew Method

Purpose

Checks whether a method is newly defined (introduced) by an emitter's target class.

IDL Syntax

boolean somtNew (
in SOMTMethodEntryC *method***);**

Note: This method does *not* take an **Environment** parameter.

Description

The **somtNew** method tests whether a method is newly defined (introduced) by an emitter's target class.

Parameters

receiver An object of class **SOMTEmitC** representing an emitter.

method An object of class **SOMTMethodEntryC** representing the method to be tested.

Return Value

The **somtNew** method returns TRUE if the specified method is introduced by the emitter's target class.

Example

```
SOMTMethodEntryC mp;
mp = _somtGetFirstMethod(__get_somtTargetClass(emitter));
if (_somtNew(emitter, mp)) /* Check to see if method is new. */
{
      _somtSetSymbolsOnEntry(mp, emitter, "method");
      _somtEmitMethod(emitter, mp);
}
```

Original Class

SOMTEmitC

Related Information

Methods: somtAll, somtInherited, somtImplemented, somtOverridden, somtNewNoProc, somtNewProc, somtVA, somtScanMethods, somtFilterNew

somtNewNoProc Method

Purpose

Checks whether a method is newly defined (introduced) by an emitter's target class and whether the method is a true method, and not a direct-call procedure.

IDL Syntax

boolean somtNewNoProc (
in SOMTEntryC *method***);**

Note: This method does *not* take an **Environment** parameter.

Description

The **somtNewNoProc** method tests whether a method is newly defined (introduced) by an emitter's target class and whether the method is a true method, and not a direct-call procedure (as determined by the absence of the "procedure" SOM IDL method modifier in the .idl file).

Parameters

receiver An object of class **SOMTEmitC** representing an emitter.

method An object of class **SOMTEntryC** representing the method to be tested.

Return Value

The **somtNewNoProc** method returns TRUE if the specified method is introduced by the emitter's target class and the method is not a direct-call procedure. Otherwise, it returns FALSE.

Example

```
SOMTClassEntryC cls = __get_somtTargetClass(emitter);
SOMTMethodEntryC method;
method = _somtGetFirstMethod(cls);
if (_somtNewNoProc(emitter, method))
   printf("Method %s is really a method!.\n",
                          __get_somtEntryName(method));
```

Original Class

SOMTEmitC

Related Information

Methods: somtOverridden, somtInherited, somtImplemented, somtNew, somtNewProc, somtOverridden, somtVA, somtScanMethods, somtFilterNew

somtNewProc Method

Purpose

Checks whether a method is newly defined (introduced) by an emitter's target class and whether the method is a direct-call procedure.

IDL Syntax

boolean somtNewProc (
in SOMTEntryC *method***);**

Note: This method does *not* take an **Environment** parameter.

Description

The **somtNewProc** method tests whether a method is newly defined (introduced) by an emitter's target class and whether the method is a direct-call procedure (i.e., it has the "procedure" SOM IDL method modifier in the .idl file).

Parameters

receiver An object of class **SOMTEmitC** representing an emitter.

method An object of class **SOMTEntryC** representing the method to be tested.

Return Value

The **somtNewProc** method returns TRUE if the specified method is introduced by the emitter's target class and the method is a direct-call procedure. Otherwise, it returns FALSE.

Example

```
SOMTClassEntryC cls = __get_somtTargetClass(emitter);
SOMTMethodEntryC method;
method = _somtGetFirstMethod(cls);
if (_somtNewProc(emitter, method))
   printf("Method %s is really a direct-call procedure.\n",
                          __get_somtEntryName(method));
```

Original Class

SOMTEmitC

Related Information

Methods: somtAll, somtInherited, somtOverridden, somtImplemented, somtNew, somtNewNoProc, somtVA, somtScanMethods, somtFilterNew

somtOpenSymbolsFile Method

Purpose

Opens the symbols file of an emitter.

IDL Syntax

FILE *somtOpenSymbolsFile (
in string *fileName***,**
in string *mode***);**

Note: This method does *not* take an **Environment** parameter.

Description

The **somtOpenSymbolsFile** method opens an emitter's symbol file (the file containing the template definitions). If the specified file does not exist in the current working directory, then the method attempts to find the file in the directories specified in the SMINCLUDE environment variable.

Parameters

receiver	An object of class **SOMTEmitC** representing an emitter.
fileName	A **string** representing the name of the file to be opened.
mode	A **string** indicating the mode in which the file should be opened (usually "r" for read only). This parameter is passed to the standard C library **fopen** function.

Return Value

A pointer to the open file is returned. If the file is not found, NULL is returned.

Example

```
FILE *fp;
SOMTTemplateOutputC template = __get_somtTemplate(emitter);
fp = _somtOpenSymbolsFile(emitter, "myfile.efw", "r");
_somtReadSectionDefinitions(template, fp);
fclose(fp);
```

Original Class

SOMTEmitC

Related Information

Methods: somtReadSectionDefinitions

somtOverridden Method

Purpose

Checks whether the specified method is an overriding method of the target class of an emitter.

IDL Syntax

boolean somtOverridden (
in SOMTMethodEntryC *method***);**

Note: This method does *not* take an **Environment** parameter.

Description

The **somtOverridden** method checks whether *method* represents an overriding method of the target class of the emitter on which the method is invoked.

Parameters

receiver An object of class **SOMTEmitC** representing an emitter.

method An object of class **SOMTMethodEntryC** representing the method to be tested.

Return Value

The **somtOverridden** method returns TRUE if the emitter's target class overrides the specified method. Otherwise, it returns FALSE.

Example

```
SOMTClassEntryC cls = __get_somtTargetClass(emitter);
SOMTMethodEntryC method;
method = _somtGetFirstMethod(cls);
if (_somtOverridden(emitter, method))
   printf("Method %s is an overriding method.\n",
                            __get_somtEntryName(method));
```

Original Class

SOMTEmitC

Related Information

Methods: somtAll, somtInherited, somtImplemented, somtNew, somtNewNoProc, somtNewProc, somtVA, somtScanMethods, somtFilterOverridden

somtScan<*Section*> **Methods**

Purpose

These methods emit a particular repeating section of an emitter's template for each item to which that section applies, preceded by the appropriate prolog section and followed by the appropriate epilog section.

IDL Syntax

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Note: These methods do *not* take an **Environment** parameter.

Description

The **somtScan**<*Section*> *scanning* methods iterate through a repeating section for each item of the emitter's target class/module to which the repeating section applies. For instance, the **somtScanBases** method iterates through the base classes of the emitter's target class, calling the section-emitting method whose name is specified by *each* for each one.

Prior to emitting the first repeating section, the specified prolog-emitting method (*prolog*) is invoked. After emitting the final repeating section, the specified epilog-emitting method (*epilog*)

is called. Methods whose names are suitable for passing as values of the *prolog, each,* and *epilog* parameters are the **somtEmit**<*Section*> methods provided by the Emitter Framework; user-written methods having the same signature as those methods can also be used.

For scanning methods that have a *filter* parameter, a section is emitted only for entries that satisfy the specified *filter* method. The following methods are suitable for passing as a *filter*: **somtNew, somtImplemented, somtOverridden, somtInherited, somtAll, somtNewProc, somtNewNoProc,** and **somtVA.** User-written methods having the same signature as those methods can also be used. For methods that have a *forceProlog* parameter, if *forceProlog* is FALSE, the prolog and epilog sections are emitted only if there is at least one entry that satisfies the specified *filter* method.

The **somtScanPassthru** method only emits sections for passthru items for which the **somtIsBeforePassthru** method returns the same value as given for **somtScanPassthru**'s *before* parameter.

The **somtScanBases, somtScanBasesF, somtScanAttributes, somtScanMethods, somtScanData, somtScanDataF,** and **somtScanPassthru** methods must only be invoked on an emitter whose target class is not NULL. The **somtScanInterfaces** and **somtScanModules** methods must only be invoked on an emitter whose target module is not NULL.

Parameters

receiver	An object of class **SOMTEmitC** representing an emitter.
prolog	A **string** representing the name of a prolog-emitting method.
each	A **string** representing the name of a repeating-section emitting method.
epilog	A **string** representing the name of an epilog-emitting method.
filter	A **string** representing the name of a filter method.
forceProlog	A **boolean** indicating whether or not to emit the prolog and epilog sections if the emitter's target class/module has no entries that satisfy the specified filter.
before	A **boolean** indicating whether to emit passthru sections for "before" passthru items (TRUE) or for "after" passthru items (FALSE).

Return Value

The **somtScan**<*Section*> methods return TRUE**.**

Example

To scan the new methods of a class:

```
_somtScanMethods(emitter,
                 "somtNew",
                 "somtEmitMethodsProlog",
                 "somtEmitMethod",
                 "somtEmitMethodsEpilog",
                 FALSE);
```

Original Class

SOMTEmitC

Related Information

Methods: somtEmit<*Section*>*,* **somtNew, somtImplemented, somtOverridden, somtInherited, somtAll, somtNewProc, somtNewNoProc, somtVA**

somtSetPredefinedSymbols Method

Purpose

Sets predefined symbols used for section names.

IDL Syntax

void somtSetPredefinedSymbols ( );

Note: This method does *not* take an **Environment** parameter.

Description

The **somtSetPredefinedSymbols** method sets predefined symbols used for section names. These symbols are used by the section-emitting methods of **SOMTEmitC** to determine which section to emit. For example, the **somtEmitProlog** method uses the value of the “prologSN” symbol to determine what section to emit. The **somtSetPredefinedSymbols** method sets the “prologSN” symbol to the value “prologS,” so that the **somtEmitProlog** method by default emits the “prologS” section.

Parameters

receiver A pointer to an object of class **SOMTEmitC** representing an emitter.

Return Value

None.

Original Class

SOMTEmitC

Related Information

Methods: somtFileSymbols, somtEmit<*Section*>

somtVA Method

Purpose

Checks whether a method accepts a variable number of arguments.

IDL Syntax

boolean somtVA (
in SOMTEntryC *method***);**

Note: This method does *not* take an **Environment** parameter.

Description

The **somtVA** method checks whether the specified method is a varargs method (that is, whether it accepts a variable number of arguments).

Parameters

receiver An object of class **SOMTEmitC** representing an emitter.

method An object of class **SOMTEntryC** representing the method to be tested.

Return Value

The **somtVA** method returns TRUE if the specified method accepts a variable number of arguments. Otherwise, it returns FALSE.

Example

```
SOMTClassEntryC cls = __get_somtTargetClass(emitter);
SOMTMethodEntryC method;
method = _somtGetFirstMethod(cls);
if (_somtVA(emitter, method))
   printf("Method %s takes a variable number of arguments.\n",
                            __get_somtEntryName(method));
```

Original Class

SOMTEmitC

Related Information

Methods: somtOverridden, somtInherited, somtImplemented, somtNew, somtNewProc, somtNewNoProc, somtOverridden, somtScanMethods

SOMTEntryC Class

Description

The SOM Compiler compiles a class interface definition (in IDL) to produce a graph structure whose nodes are instances of **SOMTEntryC** or its subclasses. Each entry (each node in the graph structure) is derived from some portion of the IDL definition, to which the attributes defined in **SOMTEntryC** and its subclasses refer. Thus, a **SOMTEntryC** object serves to hide the syntax of the class interface definition language.

The **SOMTEntryC** class provides methods for accessing information about particular portions of a class definition, such as the line number on which it occurs, its accompanying comment, its SOM IDL modifiers, its name (both scoped and unscoped), and the kind of entity represented.

File Stem

scentry

Base

SOMObject

Metaclass

SOMClass

Ancestor Classes

SOMObject

Attributes

Listed below is each available attribute, with its corresponding type in parentheses, followed by a description of its purpose:

somtEntryName (string)
The (unscoped) name of the entry**.** This attribute has no “Set” method.

somtIDLScopedName (string)
The scoped name of the entry, in IDL form (using “::” as the delimiter)**.** This attribute has no “Set” method.

somtCScopedName (string)
The scoped name of the entry, in C form (with underscore as the delimiter). This attribute has no “set” method.

somtElementType (SOMTTypes)
The type of the entry (class, method, attribute, typedef, etc.). This attribute has no “Set” method.

somtElementTypeName (string)
The string form of **somtElementType**. This attribute has no “set” method.

somtEntryComment (string)
The comment associated with the entry, or NULL if it has none. Comments will have comment delimiters removed, but will retain newline characters. This attribute has no “Set” method.

somtSourceLineNumber (unsigned long)
The line number in the source file where this entry’s syntactic form *ends*. This attribute has no “Set” method.

somtTypeCode (TypeCode)
The type code, if appropriate, otherwise NULL. This attribute has no “Set” method.

somtIsReference (boolean)
Whether this entry represents just a reference to a type, rather than a declaration of it. This attribute has no “Set” method.

New Methods

somtGetModifierValue
somtGetFirstModifier
somtGetNextModifier
somtFormatModifier
somtGetModifierList
somtSetSymbolsOnEntry

Overriding Methods

somInit
somUninit
somPrintSelf
somDumpSelfInt
somDumpSelf

somtFormatModifier Method

Purpose

Formats a SOM IDL modifier name/value pair into a buffer.

IDL Syntax

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

Note: This method does *not* take an **Environment** parameter.

Description

The **somtFormatModifier** method formats the specified SOM IDL modifier *name/value* pair into the specified *buffer*. The buffer must be large enough to hold the formatted pair; no checks are made to ensure that it is large enough. The method returns the number of characters stored in the buffer, not including the terminating NULL character.

The **somtFormatModifier** method is not intended to be invoked directly, but it can be overridden by subclasses of **SOMTEntryC** to control the format returned by the method **somtGetModifierList**.

Parameters

receiver	An object of class **SOMTEntryC**.
buffer	The address of a character buffer where the formatted output will be stored.
name	A character string representing the modifier name.
value	A character string representing the modifier value.

Return Value

The **somtFormatModifier** method stores the formatted modifier name/value pair in the specified buffer and returns the number of characters stored in the buffer, not including the terminating NULL character.

Original Class

SOMTEntryC

Related Information

Methods: somtGetModifierValue, somtGetFirstModifier, somtGetNextModifier, somtGetModifierList

somtGetFirstModifier Method

Purpose

The **somtGetFirstModifier** method gets the first modifier for a particular entry (class, attribute, method, etc.).

IDL Syntax

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

Note: This method does *not* take an **Environment** parameter.

Description

The **somtGetFirstModifier** method gets the first modifier for the entry specified by *receiver*, if it has one. Otherwise, it returns FALSE. The next modifier can be obtained using the corresponding **somtGetNextModifier** method.

Note that the **somtGetFirstModifier** and **somtGetNextModifier** methods cannot be used in doubly nested loops. For example, the following doubly nested loop will not work, because following the first execution of the inner loop, the invocation of **somtGetNextModifier** in the outer loop will return FALSE:

```
for (m1 = _somtGetFirstModifier(cls, &name, &value); m1;
     m1 = _somtGetNextModifier(cls, &name, &value))
   for (m2 = _somtGetFirstModifier(cls, &name, &value); m2;
        m2 = _somtGetNextModifier(cls, &name, &value))
    /* etc. */
```

Nested loops such as the one above are permissible if the target object (e.g., “cls”) of the inner loop differs from the target object of the outer loop, or if a different **somtGetFirst**<item> method is used in the inner loop (for instance, a **somtGetNextParameter** loop can be nested inside a **somtGetNextModifier** loop).

The **somtGetFirstModifier** method accesses the first SOM IDL modifier of the specified entry. The address of a buffer containing the name of the modifier is stored in the location pointed to by *modifierName***,** and the address of a buffer containing the value of the modifier is stored in the location pointed to by *modifierValue***.**

Parameters

receiver The entry whose first modifier is to be retrieved.

modifierName A pointer to a location where the address of a buffer containing the modifier name will be placed.

modifierValue A pointer to a location where the address of a buffer containing the modifier value will be placed.

Return Value

This method returns TRUE if a modifier name/value pair was retrieved, or FALSE if the specified entry has no modifiers.

Example

To iterate through the modifiers of a class:

```
boolean done;
string name, value;

printf("List of modifiers:\n");
for (done = _somtGetFirstModifier(cls, &name, &value); done;
    done = _somtGetNextModifier(cls, &name, &value))
    printf(" modifier %s has value %s.\n", name, value);
```

Related Information

Methods: somtGetNextModifier

somtGetModifierList Method

Purpose

Gets the SOM IDL modifiers for an entry.

IDL Syntax

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

Note: This method does *not* take an **Environment** parameter.

Description

The **somtGetModifierList** method stores the SOM IDL modifiers for the specified **SOMTEntryC** object in the specified *buffer*, separated by newline characters. The buffer must be large enough to hold all the modifiers; no checks are made to ensure that the buffer is large enough. The method returns the number of modifiers stored in the buffer.

Parameters

receiver — An object of class **SOMTEntryC** representing the entry whose modifiers are needed.

buffer — A pointer to a character buffer where the modifier list will be stored.

Return Value

The **somtGetModifierList** method stores the modifiers for the specified **SOMTEntryC** object in the specified *buffer* and returns the number of modifiers stored in the buffer.

Original Class

SOMTEntryC

Related Information

Methods: somtGetModifierValue, somtGetFirstModifier, somtGetNextModifier, somtFormatModifier

somtGetModifierValue Method

Purpose

Gets the value of a SOM IDL modifier for an entry.

IDL Syntax

string somtGetModifierValue (
in string *modifierName***);**

Note: This method does *not* take an **Environment** parameter.

Description

The **somtGetModifierValue** method returns the value of the SOM IDL modifier named *modifierName* if the entry has that modifier in the .idl file. Otherwise, it returns NULL.

Parameters

receiver An object of class **SOMTEntryC** representing the entry whose modifier is needed.

modifierName A string representing the name of the modifier whose value is needed.

Return Value

The **somtGetModifierValue** method returns a **string** representing the value of the specified modifier, if the receiver has that modifier. Otherwise, it returns NULL. If the modifier is present but has no value, then a non-NULL value is returned.

Original Class

SOMTEntryC

Related Information

Methods: somtGetFirstModifier, somtGetNextModifier, somtFormatModifier, somtGetModifierList

somtGetNextModifier Method

Purpose

This method gets the next modifier for a particular entry (class, attribute, method, etc.), relative to the previous call for a modifier.

IDL Syntax

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

Note: This method does *not* take an **Environment** parameter.

Description

The **somtGetNextModifier** methods return the next modifier for the entry on which the method was invoked, if it has a next modifier. Otherwise, it returns FALSE.

A call to a **somtGetNextModifier** method is relative to the last call of either the same method or the corresponding **somtGetFirstModifier** method**,** applied to the same entry object. Note that this implies that the **somtGetFirstModifier** and **somtGetNextModifier** methods cannot be used in doubly nested loops. For example, the following doubly nested loop will not work, because following the first execution of the inner loop, the invocation of **somtGetNextModifier** in the outer loop will return FALSE:

```
for (m1 = _somtGetFirstModifier(cls, &name, &value); m1;
     m1 = _somtGetNextModifier(cls, &name, &value))
   for (m2 = _somtGetFirstModifier(cls, &name, &value); m2;
        m2 = _somtGetNextModifier(cls, &name, &value))
    /* etc. */
```

Nested loops such as the one above are permissible if the target object (e.g., “cls”) of the inner loop differs from the target object of the outer loop, or if a different **somtGetFirst**<item> method is used in the inner loop (for instance, a **somtGetNextParameter** loop can be nested inside a **somtGetNextModifier** loop).

The **somtGetNextModifier** method accesses the next SOM IDL modifier of the specified entry, relative to the last call to **somtGetFirstModifier** or **somtGetNextModifier** on the same entry. The address of a buffer containing the name of the modifier is stored in the location pointed to by *modifierName***,** and the address of a buffer containing the value of the modifier is stored in the location pointed to by *modifierValue***.** If the specified entry had at least one more modifier, the method returns TRUE. Otherwise, it returns FALSE.

Parameters

receiver The entry whose next modifier is to be retrieved.

modifierName A pointer to a location where the address of a buffer containing the modifier name will be placed.

modifierValue A pointer to a location where the address of a buffer containing the modifier value will be placed.

Return Value

This method returns TRUE if a modifier name/value pair was retrieved, or FALSE is the specified entry has no next modifier.

Example

To iterate through the modifiers of a class:

```
boolean done;
string name, value;

printf("List of modifiers:\n");
for (done = _somtGetFirstModifier(cls, &name, &value); done;
    done = _somtGetNextModifier(cls, &name, &value))
    printf(" modifier %s has value %s.\n", name, value);
```

Related Information

Methods: somtGetFirstModifier

somtSetSymbolsOnEntry Method

Purpose

Places predefined symbol/value pairs for an entry into the symbol table.

IDL Syntax

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

Note: This method does *not* take an **Environment** parameter.

Description

The **somtSetSymbolsOnEntry** method places predefined symbol/value pairs for the specified entry in the specified emitter's symbol table. In the default implementation, all symbol names begin with the string specified in *prefix* (e.g., "class," "method," "data," "passthru"). For example, **SOMTEntryC**'s implementation of **somtSetSymbolsOnEntry** defines the <prefix>Name and <prefix>Comment symbols. Hence, when invoked on a **SOMTClassEntryC** object with *prefix* = "class," this method defines symbols "className" and "classComment" for that entry.

This method is invoked by the **somtEmit**<*Section*> methods that take an entry as an additional parameter, so that symbols will be defined for that entry when the section is emitted. For example, the **somtEmitBase** method invokes **somtSetSymbolsOnEntry** on the **SOMTBaseClassEntryC** object passed to it (using a prefix of "base") prior to emitting the "baseS" section, so that the symbols baseName, baseComment, baseLineNumber, etc., used within the "baseS" section will have values appropriate for that base-class entry.

This method can be overridden in subclasses of **SOMTEntryC** (or subclasses of any of its subclasses) to define additional symbols or to change the default symbol settings. Overriding methods should invoke the parent method either before or after defining new symbols.

Parameters

receiver	An object of class **SOMTEntryC** for which symbols are to be defined.
emitter	An object of class **SOMTEmitC** representing an emitter.
prefix	A **string** representing the symbol-name prefix to be used.

Return Value

The **somtSetSymbolsOnEntry** method returns 1.

Example

```
SOMTMethodEntryC method;
SOMTClassEntryC class = __get_somtTargetClass(emitter);
SOMTTemplateOutputC template = __get_somtTemplate(emitter);
for (method = _somtGetFirstMethod(class); method;
     method = _somtGetNextMethod(class))
{
   _somtSetSymbolsOnEntry(method, emitter, "method");
   _somtOutputSection(template, "myMethodSection");
}
```

Original Class

SOMTEntryC

Related Information

Method: somtExpandSymbol, somtCheckSymbol, somtSetSymbolCopyBoth, somtSetSymbolCopyValue, somtGetSymbol, somtSetSymbol, somtSetSymbolCopyName

SOMTEnumEntryC Class

Description

A **SOMTEnumEntryC** object represents an enumeration definition. It provides methods for accessing the enumerator names within the enumeration.

File Stem

scenum

Base

SOMTEntryC

Metaclass

SOMClass

Ancestor Classes

SOMTEntryC, **SOMObject**

Attributes

None.

New Methods

somtGetFirstEnumName
somtGetNextEnumName

Overriding Methods

somtSetSymbolsOnEntry
somDumpSelfInt

somtGetFirstEnumName Method

Purpose

Gets the first enumerator name for a **SOMTEnumEntryC** object.

IDL Syntax

SOMTEnumNameEntryC somtGetFirstEnumName ( );

Note: This method does *not* take an **Environment** parameter.

Description

The **somtGetFirstEnumName** method returns the first enumerator name for the entry on which the method was invoked. The next enumerator name can be obtained using the corresponding **somtGetNextEnumName** method.

Note that the **somtGetFirstEnumName** and **somtGetNextEnumName** methods cannot be used in doubly nested loops. For example, the following doubly nested loop will not work, because following the first execution of the inner loop, the invocation of **somtGetNextEnumName** in the outer loop will return NULL:

```
for (e1 = _somtGetFirstEnumName(enum); e1;
     e1 = _somtGetNextEnumName(enum))
   for (e2 = _somtGetFirstEnumName(enum); e2;
        e2 = _somtGetNextEnumName(enum))
    /* etc. */
```

Parameters

receiver The enumeration entry whose first enumerator name is to be retrieved.

Return Value

This method returns the first enumerator name for the specified enumeration entry.

Example

To iterate through the enumerator names of an enumeration entry:

```
SOMTEnumNameEntryC myEntry;

printf("List of enumerator names:\n");
for (myEntry = _somtGetFirstEnumName(enum); myEntry;
    myEntry = _somtGetNextEnumName(enum))
    printf("%s\n", __get_somtEntryName(myEntry));
```

Related Information

Methods: somtGetNextEnumName

somtGetNextEnumName Method

Purpose

Gets the next enumerator name for an enumeration entry, relative to the previous call for an enumerator name.

IDL Syntax

SOMTEnumNameEntryC somtGetNextEnumName ( );

Note: This method does *not* take an **Environment** parameter.

Description

The **somtGetNextEnumName** method returns the next enumerator name for the entry on which the method was invoked, if it has a next enumerator name. Otherwise, it returns NULL.

A call to a **somtGetNextEnumName** method is relative to the last call of either the same method or the corresponding **somtGetFirstEnumName** method**,** applied to the same entry object. Note that this implies that the **somtGetFirstEnumName** and **somtGetNextEnumName** methods cannot be used in doubly nested loops. For example, the following doubly nested loop will not work, because following the first execution of the inner loop, the invocation of **somtGetNextEnumName** in the outer loop will return NULL:

```
for (e1 = _somtGetFirstEnumName(enum); e1;
     e1 = _somtGetNextEnumName(enum))
   for (e2 = _somtGetFirstEnumName(enum); e2;
        e2 = _somtGetNextEnumName(enum))
    /* etc. */
```

Parameters

receiver The entry whose next enumerator name is to be retrieved.

Return Value

This method returns the next enumerator name for the entry represented by *receiver*, if it has a next enumerator. Otherwise, it returns NULL.

Example

To iterate through the enumerator names of an enumeration entry:

```
SOMTEnumNameEntryC myEntry;

printf("List of enumerator names:\n");
for (myEntry = _somtGetFirstEnumName(enum); myEntry;
    myEntry = _somtGetNextEnumName(enum))
    printf("%s\n", __get_somtEntryName(myEntry));
```

Related Information

Methods: somtGetFirstEnumName

SOMTEnumNameEntryC Class

Description

A **SOMTEnumNameEntryC** object represents an enumerator name. It provides attributes for accessing the enumeration in which it is defined and its value.

File Stem

scenumnm

Base

SOMTEntryC

Metaclass

SOMClass

Ancestor Classes

SOMTEntryC, **SOMObject**

Attributes

Listed below is each available attribute, with its corresponding type in parentheses, followed by a description of its purpose:

somtEnumPtr (SOMTEnumEntryC)
A pointer to the enumeration that defines this enumerator name. This attribute has no “set” method.

somtEnumVal (unsigned long)
The value of the enumerator. This attribute has no “set” method.

New Methods

None.

Overriding Methods

somtSetSymbolsOnEntry
somDumpSelfInt

SOMTMetaClassEntryC Class

Description

A **SOMTMetaClassEntryC** object represents the metaclass statement in the implementation section of a SOM IDL class interface definition. The actual metaclass is represented by a **SOMTClassEntryC** object accessed via the **somtMetaClassDef** attribute.

File Stem

scmeta

Parent

SOMTEntryC

Metaclass

SOMClass

Ancestor Classes

SOMTEntryC, **SOMObject**

Attributes

Listed below is each available attribute, with its corresponding type in parentheses, followed by a description of its purpose:

somtMetaFile (string)
The name of the file containing the definition of this metaclass.
This attribute has no “Set” method.

somtMetaClassDef (SOMTClassEntryC)
A pointer to an entry object representing the class definition entry for this metaclass.
This attribute has no “Set” method.

New Methods

None.

Overriding Methods

somtSetSymbolsOnEntry,
somDumpSelfInt

SOMTMethodEntryC Class

Description

A **SOMTMethodEntryC** object represents a method declaration within a class interface definition. It provides attributes and methods for accessing the method's type, arguments, context, exceptions, and parameters. For overriding methods, it also provides attributes for accessing the overridden method and the overridden method's class.

File Stem

scmethod

Base

SOMTCommonEntryC

Metaclass

SOMClass

Ancestor Classes

SOMTCommonEntryC, **SOMTEntryC**, **SOMObject**

Attributes

Listed below is each available attribute, with its corresponding type in parentheses, followed by a description of its purpose:

somtCReturnType (string)
The C/C++ return type of the method.
This attribute has no "Set" method.

somtIsVarargs (boolean)
TRUE if the method definition includes a **va_list** parameter, and FALSE otherwise.
This attribute has no "Set" method.

somtOriginalMethod (SOMTMethodEntryC)
For an overriding method definition, the entry for the method being overridden.
This attribute has no "Set" method.

somtOriginalClass (SOMTClassEntryC)
For an overriding method, the entry for the class whose method is being overridden.
For a new method, the entry for the class that introduced the method.
This attribute has no "Set" method.

somtIsOneway (boolean)
Whether the method is defined as oneway. This attribute has no "Set" method.

somtArgCount (short)
The number of *explicit* arguments of the method (not including the method's receiver, the **Environment** parameter, or the **Context** parameter, if any).
This attribute has no "Set" method.

somtContextArray (string *)
An array of the context string–literals for the method (given in the "context" expression in the method's IDL definition). This attribute has no "set" method.

New Methods

somtGetFirstParameter
somtGetNextParameter
somtGetIDLParamList
somtGetShortCParamList
somtGetFullCParamList
somtGetShortParamNameList
somtGetFullParamNameList
somtGetNthParameter
somtGetFirstException
somtGetNextException

Overriding Methods

somtSetSymbolsOnEntry
somDumpSelfInt

somtGetFirst<*Item*> Methods

Purpose

These methods get the first exception or parameter for a method entry.

IDL Syntax

SOMTStructEntryC somtGetFirstException ( );

SOMTParameterEntryC somtGetFirstParameter ( );

Note: These methods do *not* take an **Environment** parameter.

Description

The **somtGetFirst**<*Item*> methods return the first exception or parameter for the entry on which the method was invoked, if it has one. Otherwise, it returns NULL. The next item of the same kind can be obtained using the corresponding **somtGetNext**<*Item*> method. For example, the **somtGetFirstException** method returns the entry representing the first exception of the specified method. If the method has no exceptions, it returns NULL. The **somtGetNextException** method can be used repeatedly to retrieve each successive exception.

Note that the same **somtGetFirst**<*Item*> and **somtGetNext**<*Item*> methods cannot be used in doubly nested loops. For example, the following doubly nested loop will not work, because following the first execution of the inner loop, the invocation of **somtGetNextException** in the outer loop will return NULL:

```
for (e1 = _somtGetFirstException(method); e1;
     e1 = _somtGetNextException(method))
   for (e2 = _somtGetFirstException(method); e2;
        e2 = _somtGetNextException(method))
    /* etc. */
```

Nested loops such as the one above are permissible if the target object (e.g., “method”) of the inner loop differs from the target object of the outer loop, or if a different **somtGetFirst**<*Item*> method is used in the inner loop (for instance, a **somtGetNextParameter** loop can be nested inside a **somtGetNextException** loop).

Note that for the **somtGetFirstParameter** method, only explicit method parameters are returned. In other words, the method’s *receiver*, the **Environment** parameter, and the **Context** parameter are not included as parameters.

Parameters

receiver The method entry whose first item is to be retrieved.

Return Value

These methods return the first exception or parameter for a method entry. The type of item returned is specific to the method; see the method call syntax shown above.

Example

To iterate through the parameters of a method:

```
SOMTParameterEntryC myEntry;

printf("List of parameters:\n");
for (myEntry = _somtGetFirstParameter(method); myEntry;
     myEntry = _somtGetNextParameter(method))
     printf("%s\n", __get_somtEntryName(myEntry));
```

Related Information

Methods: somtGetNext<*Item*>

somtGetFullCParamList Method

Purpose

Gets the formal parameter list, including the type and name of each parameter, of a method's C procedure.

IDL Syntax

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

Note: This method does *not* take an **Environment** parameter.

Description

The **somtGetFullCParamList** method returns the formal parameter list for the specified method's C procedure. The list includes both the type and the name of each parameter (unlike **somtGetShortParamNameList**, which includes only the parameter names). The parameter list is built in *buffer* and the address of *buffer* is returned. The list is delimited by newlines rather than commas so that it can be used as a symbol value suitable for list substitution in the template.

Unlike the **somtGetIDLParamList** method, this method gives the types of the parameters as they appear in the method's C procedure, which may differ from the IDL types. For example, an out or inout parameter may have pointer stars added in the C form. Unlike the method **somtGetShortCParamList,** this method *does* include in the result the method's receiver, the **Environment** parameter, or the **Context** parameter, if any.

If the method takes a variable number of arguments, then the va_list parameter is replaced by the string specified in *varargsParm*, unless *varargsParm* is NULL, in which case the va_list parameter is removed.

Parameters

receiver — An object of class **SOMTMethodEntryC** representing a method entry whose parameter list is needed.

buffer — The address of a character buffer to receive the argument list.

varargsParm — A **string** representing the variable arguments parameter, or NULL to remove the variable arguments parameter.

Return Value

The **somtGetFullCParamList** method returns a string representing the formal argument list of the method's C procedure, including the type and name of each argument. This string is stored in *buffer*. The list is delimited by newlines rather than commas so that it can be used as a symbol value suitable for list substitution in the template.

Example

```
char buf[MAX_BUFFER];
SOMTMethodEntryC method;
SOMTClassEntryC class = __get_somtTargetClass(emitter);
SOMTTemplateOutputC template = __get_somtTemplate(emitter);
method = _somtGetFirstMethod(class);
_somtGetFullCParamList(method, buf, "va_list ap");
_somtSetSymbolCopyBoth(template, "methodFullCParmList", buf);
```

Original Class

SOMTMethodEntryC

Related Information

Methods: somtGetFirstParameter, **somtGetNextParameter**, **somtGetIDLParamList, somtGetShortCParamList, somtGetShortParamNameList, somtGetFullParamNameList, somtGetNthParameter**

somtGetFullParamNameList Method

Purpose

Gets the list of parameter names for a method's procedure.

IDL Syntax

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

Note: This method does *not* take an **Environment** parameter.

Description

The **somtGetFullParamNameList** method returns a list of parameter names (but not their corresponding types, as for **somtGetFullCParamList**) for a specified method's procedure. Unlike the **somtGetShortParamNameList,** the method's receiver, the **Environment** parameter, and the **Context** parameter, if present, *are* included in the list.

The argument list is built in *buffer* and the address of *buffer* is returned. The list is delimited by newlines rather than commas so that it can be used as a symbol value suitable for list substitution in the template.

If the method takes a variable number of arguments, then the va_list parameter is replaced by the string specified by *varargsParm*, unless *varargsParm* is NULL, in which case the va_list parameter is removed.

Parameters

receiver	A **SOMTMethodEntryC** object representing a method whose parameter names are needed.
buffer	The address of a buffer in which to build the parameter list.
varargsParm	A **string** representing the variable arguments parameter, or NULL to remove the variable arguments parameter.

Return Value

The **somtGetFullParamNameList** returns a pointer to the string representing the parameter list for the specified method's procedure.

Example

To get the parameter list of a method's procedure:

```
char buf[MAX_BUFFER];
SOMTMethodEntryC method;
SOMTClassEntryC class = __get_somtTargetClass(emitter);
SOMTTemplateOutputC template = __get_somtTemplate(emitter);
method = _somtGetFirstMethod(class);
_somtGetFullParamNameList(method, buf, "ap");
_somtSetSymbolCopyBoth(template, "methodFullParmNameList", buf);
```

Original Class

SOMTMethodEntryC

Related Information

Methods: somtGetFirstParameter, **somtGetNextParameter**, **somtGetShortCParamList, somtGetFullCParamList, somtGetIDLParamList, somtGetShortParamNameList, somtGetNthParameter**

somtGetIDLParamList Method

Purpose

Gets the formal parameter list, including the type and name of each parameter, of a method, in IDL syntax.

IDL Syntax

string somtGetIDLParamList (
in string *buffer***);**

Note: This method does *not* take an **Environment** parameter.

Description

The **somtGetIDLParamList** method returns the formal parameter list (in IDL form) for the specified method. The list includes both the type and the name of each parameter; for example, `"in int x\n in float y\n in char z"`. The parameter list is built in *buffer* and the address of *buffer* is returned. The list is delimited by newlines rather than commas so that it can be used as a symbol value suitable for list substitution in the template.

Parameters

receiver — An object of class **SOMTMethodEntryC** representing a method entry whose parameter list is needed.

buffer — The address of a character buffer to receive the parameter list.

Return Value

The **somtGetIDLParamList** method returns a string representing the formal parameter list of the method, including the type and name of each parameter, in IDL syntax. This string is stored in buffer. The list is delimited by newlines rather than commas so that it can be used as a symbol value suitable for list substitution in the template.

Example

```
char buf[MAX_BUFFER];
SOMTMethodEntryC method;
SOMTClassEntryC class = __get_somtTargetClass(emitter);
SOMTTemplateOutputC template = __get_somtTemplate(emitter);
method = _somtGetFirstMethod(class);
_somtGetIDLParamList(method, buf);
_somtSetSymbolCopyBoth(template, "methodIDLParmList", buf);
```

Original Class

SOMTMethodEntryC

Related Information

Methods: somtGetFirstParameter, **somtGetNextParameter**, **somtGetShortCParamList, somtGetFullCParamList, somtGetShortParamNameList, somtGetFullParamNameList, somtGetNthParameter**

somtGetNext<*Item*> Methods

Purpose

These methods get the next exception or parameter for a method entry, relative to the previous call for a similar entry.

IDL Syntax

SOMTStructEntryC somtGetNextException ( );

SOMTParameterEntryC somtGetNextParameter ( );

Note: These methods do *not* take an **Environment** parameter.

Description

The **somtGetNext**<*Item*> methods return the next item (of the type shown above) for the entry on which the method was invoked, if it has a next item of that type. Otherwise, it returns NULL.

A call to a **somtGetNext**<*Item*> method is relative to the last call of either the same method or the corresponding **somtGetFirst**<*Item*> method**,** applied to the same entry object. Note that this implies that the same **somtGetFirst**<*Item*> and **somtGetNext**<*Item*> methods cannot be used in doubly nested loops. For example, the following doubly nested loop will not work, because following the first execution of the inner loop, the invocation of **somtGetNextException** in the outer loop will return NULL:

```
for (e1 = _somtGetFirstException(method); e1;
     e1 = _somtGetNextException(method))
   for (e2 = _somtGetFirstException(method); e2;
        e2 = _somtGetNextException(method))
    /* etc. */
```

Nested loops such as the one above are permissible if the target object (e.g., “method”) of the inner loop differs from the target object of the outer loop, or if a different **somtGetFirst**<*Item*> method is used in the inner loop (for instance, a **somtGetNextParameter** loop can be nested inside a **somtGetNextMethod** loop).

Note that for the **somtGetNextParameter** method, only explicit method parameters are returned. In other words, the method’s *receiver*, the **Environment** parameter, and the **Context** parameter are not included as parameters.

Parameters

receiver The method entry whose next item is to be retrieved.

Return Value

These methods return the next exception or parameter for the entry on which the method was invoked, if it has a next item of that type. Otherwise, it returns NULL. The type of item returned is specific to the method; see the method call syntax shown above.

Example

To iterate through the parameters of a method:

```
SOMTParameterEntryC myEntry;

printf("List of parameters:\n");
for (myEntry = _somtGetFirstParameter(method); myEntry;
     myEntry = _somtGetNextParameter(method))
     printf("%s\n", __get_somtEntryName(myEntry));
```

Related Information

Methods: somtGetFirst<*Item*>

somtGetNthParameter Method

Purpose

Gets the entry representing a particular parameter of a method.

IDL Syntax

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

Note: This method does *not* take an **Environment** parameter.

Description

The **somtGetNthParameter** method returns the entry object representing the specified explicit parameter of the *receiver*. The first argument is numbered zero. Note that the receiver of the method, the **Environment** parameter, and the **Context** parameter, if any, are not returned and are not counted.

Parameters

receiver — A **SOMTMethodEntryC** object representing a method whose parameter is needed.

n — The number of the parameter to return, starting from zero.

Return Value

The **somtGetNthParameter** returns the **SOMTParameterEntryC** object representing the *n*th parameter of *receiver*.

Original Class

SOMTMethodEntryC

Related Information

Methods: somtGetFirstParameter, **somtGetNextParameter**, **somtGetShortCParamList, somtGetFullCParamList, somtGetIDLParamList, somtGetFullParamNameList, somtGetShortParamNameList**

somtGetShortCParamList Method

Purpose

Gets the formal parameter list, including the type and name of each parameter, of a method's C procedure.

IDL Syntax

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

Note: This method does *not* take an **Environment** parameter.

Description

The **somtGetShortCParamList** method returns the formal parameter list for the specified method's C procedure. The list includes both the type and the name of each parameter (unlike **somtGetShortParamNameList**, which includes only the parameter names). The parameter list is built in *buffer* and the address of *buffer* is returned. The list is delimited by newlines rather than commas so that it can be used as a symbol value suitable for list substitution in the template.

Unlike the **somtGetIDLParamList** method, this method gives the types of the parameters as they appear in the method's C procedure, which may differ from the IDL types. For example, an out or inout parameter may have pointer stars added in the C form. Unlike the method **somtGetFullCParamList,** this method does *not* include in the result the method's receiver, the **Environment** parameter, or the **Context** parameter, if any.

If *selfParm* is not null, then it is added as an initial parameter. The *selfParm* value can contain multiple parameters, delimited by newlines.

If the method takes a variable number of parameters, then the va_list parameter is replaced by the string specified by *varargsParm*, unless *varargsParm* is NULL, in which case the va_list parameter is removed.

Parameters

receiver — An object of class **SOMTMethodEntryC** representing a method entry whose parameter list is needed.

buffer — The address of a character buffer to receive the parameter list.

selfParm — A **string** representing a parameter (or list of newline-separated parameters) to be inserted at the start of the list.

varargsParm — A **string** representing the variable arguments parameter, or NULL to remove the variable arguments parameter.

Return Value

The **somtGetShortCParamList** method returns a string representing the formal parameter list of the method's C procedure, including the type and name of each parameter. This string is stored in buffer. The list is delimited by newlines rather than commas so that it can be used as a symbol value suitable for list substitution in the template.

Example

```
char buf[MAX_BUFFER];
SOMTMethodEntryC method;
SOMTClassEntryC class = __get_somtTargetClass(emitter);
SOMTTemplateOutputC template = __get_somtTemplate(emitter);
method = _somtGetFirstMethod(class);
_somtGetShortCParamList(method, buf, NULL, "va_list ap");
_somtSetSymbolCopyBoth(template, "methodShortCParmList", buf);
```

Original Class

SOMTMethodEntryC

Related Information

Methods: somtGetFirstParameter, **somtGetNextParameter**, **somtGetIDLParamList, somtGetFullCParamList, somtGetShortParamNameList, somtGetFullParamNameList, somtGetNthParameter**

somtGetShortParamNameList Method

Purpose

Gets the list of explicit parameter names for a method.

IDL Syntax

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

Note: This method does *not* take an **Environment** parameter.

Description

The **somtGetShortParamNameList** method returns a list of explicit parameter names (but not their corresponding types, as for **somtGetIDLParamList**) for a specified method. Unlike **somtGetFullParamNameList**, the method's receiver, the **Environment** parameter, and the **Context** parameter are *not* included in the list; only the explicit method parameters (as declared in IDL) are present in the result of this method.

The parameter list is built in *buffer* and the address of *buffer* is returned. The list is delimited by newlines rather than commas so that it can be used as a symbol value suitable for list substitution in the template.

If *selfParm* is not null, then it is added as an initial parameter. The *selfParm* value can contain multiple parameters, delimited by newlines.

If the method takes a variable number of arguments, then the va_list parameter is replaced by the string specified by *varargsParm*, unless *varargsParm* is NULL, in which case the va_list parameter is removed.

Parameters

receiver	A **SOMTMethodEntryC** object representing the method whose parameter names are needed.
buffer	The address of a buffer in which to build the parameter list.
selfParm	A **string** representing a parameter (or list of newline-separated parameters) to be inserted at the start of the list.
varargsParm	A **string** representing the variable arguments parameter, or NULL to remove the variable arguments parameter.

Return Value

The **somtGetShortParamNameList** returns a pointer to the string representing the parameter list for the specified method.

Example

To get the parameter list of a method:

```
char buf[MAX_BUFFER];
SOMTMethodEntryC method;
SOMTClassEntryC class = __get_somtTargetClass(emitter);
SOMTTemplateOutputC template = __get_somtTemplate(emitter);
method = _somtGetFirstMethod(class);
_somtGetShortParamNameList(method, buf, NULL, "ap");
_somtSetSymbolCopyBoth(template, "methodShortParmNameList", buf);
```

Original Class

SOMTMethodEntryC

Related Information

Methods: somtGetFirstParameter, **somtGetNextParameter**, **somtGetShortCParamList, somtGetFullCParamList, somtGetIDLParamList, somtGetFullParamNameList, somtGetNthParameter**

SOMTModuleEntryC Class

Description

A **SOMTModuleEntryC** object represents an IDL module definition. It provides methods for accessing the structs, unions, enums, types, sequences, constants, interfaces, and nested modules within the module.

File Stem

scmodule

Base

SOMTEntryC

Metaclass

SOMClass

Ancestor Classes

SOMTEntryC, **SOMObject**

Attributes

Listed below is each available attribute, with its corresponding type in parentheses, followed by a description of its purpose:

somtModuleFile (string)
The name of the file in which the module is defined.
This attribute has no “Set” method.

somtOuterModule (SOMTModuleEntryC)
The module enclosing this module, or NULL if there is none.

New Methods

somtGetFirstModuleStruct
somtGetNextModuleStruct
somtGetFirstModuleTypedef
somtGetNextModuleTypedef
somtGetFirstModuleUnion
somtGetNextModuleUnion
somtGetFirstModuleEnum
somtGetNextModuleEnum
somtGetFirstModuleConstant
somtGetNextModuleConstant
somtGetFirstModuleSequence
somtGetNextModuleSequence
somtGetFirstInterface
somtGetNextInterface
somtGetFirstModule
somtGetNextModule
somtGetFirstModuleDef
somtGetNextModuleDef

Overriding Methods

somDumpSelfInt
somtSetSymbolsOnEntry

somtGetFirst<*Item*> Methods

Purpose

The **somtGetFirst**<*Item*> methods get the first item (such as a constant, typedef, union, etc.) for a module entry.

IDL Syntax

SOMTClassEntryC somtGetFirstInterface ( );

SOMTModuleEntryC somtGetFirstModule ( );

SOMTConstEntryC somtGetFirstModuleConstant ( );

SOMTEnumEntryC somtGetFirstModuleEnum ( );

SOMTSequenceEntryC somtGetFirstModuleSequence ( );

SOMTStructEntryC somtGetFirstModuleStruct ( );

SOMTTypedefEntryC somtGetFirstModuleTypedef ( );

SOMTUnionEntryC somtGetFirstModuleUnion ( );

SOMTEntryC somtGetFirstModuleDef ( );

Note: These methods do *not* take an **Environment** parameter.

Description

The **somtGetFirst**<*Item*> methods return the first item of the type shown above for the entry on which the method was invoked, if it has one. Otherwise, it returns NULL. The next item of the same kind can be obtained using the corresponding **somtGetNext**<*Item*> method. For example, the **somtGetFirstInterface** method returns the entry representing the first interface of the specified module. If the module has no interfaces, it returns NULL. The **somtGetNextInterface** can be used repeatedly to retrieve each successive interface. The **somtGetFirstModuleDef** method returns the first constant/type definition of the module, whether a typedef, struct, union, interface name etc.

Note that the same **somtGetFirst**<*Item*> and **somtGetNext**<*Item*> methods cannot be used in doubly nested loops. For example, the following doubly nested loop will not work, because following the first execution of the inner loop, the invocation of **somtGetNextInterface** in the outer loop will return NULL:

```
for (m1 = _somtGetFirstInterface(mod); m1;
     m1 = _somtGetNextInterface(mod))
   for (m2 = _somtGetFirstInterface(mod); m2;
        m2 = _somtGetNextInterface(mod))
    /* etc. */
```

Nested loops such as the one above are permissible if the target object (e.g., “mod”) of the inner loop differs from the target object of the outer loop, or if a different **somtGetFirst**<*Item*> method is used in the inner loop (for instance, a **somtGetNextModuleConstant** loop can be nested inside a **somtGetNextInterface** loop).

Parameters

receiver The entry whose first item is to be retrieved.

Return Value

These methods return the first item (such as a constant, union, typedef, etc.) for a module entry. The type of item returned is specific to the method; see the method call syntax shown above.

Example

To iterate through the nested modules of a module:

```
SOMTModuleEntryC myEntry;

printf("List of nested modules:\n");
for (myEntry = _somtGetFirstModule(mod); myEntry;
     myEntry = _somtGetNextModule(mod))
     printf("%s\n", __get_somtEntryName(myEntry));
```

Related Information

Methods: somtGetNext<*Item*>

somtGetNext<*Item*> Methods

Purpose

These methods get the next item (such as a constant, union, typedef, etc.) for a module entry, relative to the previous call for a similar entry.

IDL Syntax

SOMTClassEntryC somtGetNextInterface ( );

SOMTModuleEntryC somtGetNextModule ( );

SOMTConstEntryC somtGetNextModuleConstant ( );

SOMTEnumEntryC somtGetNextModuleEnum ( );

SOMTSequenceEntryC somtGetNextModuleSequence ( );

SOMTStructEntryC somtGetNextModuleStruct ( );

SOMTTypedefEntryC somtGetNextModuleTypedef ( );

SOMTUnionEntryC somtGetNextModuleUnion ( );

SOMTEntryC somtGetNextModuleDef ( );

Note: These methods do *not* take an **Environment** parameter.

Description

The **somtGetNext**<*Item*> methods return the next item (of the type shown above) for the entry on which the method was invoked, if it has a next item of that type. Otherwise, it returns NULL. The **somtGetNextModuleDef** method returns the next constant/type definition of the module, whether a typedef, struct, union, interface name etc.

A call to a **somtGetNext**<*Item*> method is relative to the last call of either the same method or the corresponding **somtGetFirst**<*Item*> method**,** applied to the same entry object. Note that this implies that the same **somtGetFirst**<*Item*> and **somtGetNext**<*Item*> methods cannot be used in doubly nested loops. For example, the following doubly nested loop will not work, because following the first execution of the inner loop, the invocation of **somtGetNextInterface** in the outer loop will return NULL:

```
for (m1 = _somtGetFirstInterface(mod); m1;
     m1 = _somtGetNextInterface(mod))
   for (m2 = _somtGetFirstInterface(mod); m2;
        m2 = _somtGetNextInterface(mod))
    /* etc. */
```

Nested loops such as the one above are permissible if the target object (e.g., “mod”) of the inner loop differs from the target object of the outer loop, or if a different **somtGetFirst**<*Item*> method is used in the inner loop (for instance, a **somtGetNextModuleConstant** loop can be nested inside a **somtGetNextInterface** loop).

Parameters

receiver The entry whose next item is to be retrieved.

Return Value

These methods return the next item (of the type shown above) for the entry represented by *receiver*, if it has a next item of that type. Otherwise, it returns NULL. The type of item returned is specific to the method; see the method call syntax shown above.

Example

To iterate through the nested modules of a module:

```
SOMTModuleEntryC myEntry;

printf("List of nested modules:\n");
for (myEntry = _somtGetFirstModule(mod); myEntry;
     myEntry = _somtGetNextModule(mod))
     printf("%s\n", __get_somtEntryName(myEntry));
```

Related Information

Methods: somtGetFirst<*Item*>

SOMTParameterEntryC Class

Description

A **SOMTParameterEntryC** object represents a parameter of a method.

File Stem

scparm

Base

SOMTCommonEntryC

Metaclass

SOMClass

Ancestor Classes

SOMTCommonEntryC, **SOMTEntryC**, **SOMObject**

Types

enum somtParameterDirectionT { somtInE, somtOutE, somtInOutE }

Attributes

somtParameterDirection (somtParameterDirectionT)
The I/O direction for the parameter.
There are three possible parameter directions:
somtInE: The parameter is for input.
somtOutE: The parameter is for output.
somtInOutE The parameter is for both input and output.

somtIDLParameterDeclaration (string)
The IDL declaration of the parameter, including its type and name.

somtCParameterDeclaration (string)
The declaration for the parameter within a C/C++ method procedure prototype. This may differ from the IDL declaration; in particular, pointer stars may be added, depending on the direction of the parameter.

New Methods

None.

Overriding Methods

somtSetSymbolsOnEntry
somDumpSelfInt

SOMTPassthruEntryC Class

Description

An object of class **SOMTPassthruEntryC** represents a passthru item in a class definition. It provides attributes for accessing the target, language, and body of the passthru, as well as whether the passthru is a “before” or “after” passthru.

File Stem

scpass

Base

SOMTEntryC

Metaclass

SOMClass

Ancestor Classes

SOMTEntryC, **SOMObject**

Attributes

Listed below is each available attribute, with its corresponding type in parentheses, followed by a description of its purpose:

somtPassthruTarget (string)
The target emitter (h, xh, ih, etc.) for a passthru entry.
This attribute has no “Set” method.

somtPassthruLanguage (string)
The name of the language for which a passthru entry is intended.
Language names are always in upper case.
(“C” is used for both C and C++.”)
This attribute has no “Set” method.

somtPassthruBody (string)
The source text of a passthru entry, without modification.
Newlines present in the source are retained.

New Methods

somtIsBeforePassthru

Overriding Methods

somtSetSymbolsOnEntry
somDumpSelfInt

somtIsBeforePassthru Method

Purpose

Tests whether a passthru entry represents a “before” or “after” passthru.

IDL Syntax

boolean somtIsBeforePassthru (SOMTPassthruEntryC *receiver***);**

Note: This method does *not* take an **Environment** parameter.

Description

The **somtIsBeforePassthru** method tests whether a passthru entry represents a passthru intended to be put at the beginning of the output file or inserted after the #include statements, as specified in the .idl file.

Parameters

receiver An object of class **SOMTPassthruEntryC** representing a passthru item to be tested.

Return Value

The **somtIsBeforePassthru** method returns TRUE if this passthru entry is a “before” passthru (intended to be put at the beginning of the emitted file), or it returns FALSE if this passthru entry is an “after” passthru (intended to be put after the #include statements in the emitted file).

Original Class

SOMTPassthruEntryC

Related Information

Methods: somtEmitFullPassthru, somtGetFirstPassthru, somtGetNextPassthru, somtEmitPassthru

SOMTSequenceEntryC Class

Description

A **SOMTSequenceEntryC** object represents a sequence type definition. It provides attributes for accessing the type and length of the sequence.

File Stem

scseqnce

Base

SOMTEntryC

Metaclass

SOMClass

Ancestor Classes

SOMTEntryC, **SOMObject**

Attributes

Listed below is each available attribute, with its corresponding type in parentheses, followed by a description of its purpose:

somtSeqLength (long)
The length of the sequence, as specified in the .idl file. If unspecified, this attribute is zero. This attribute has no “set” method.

somtSeqType (SOMTEntryC)
A pointer to an object representing the type of the sequence. This attribute has no “set” method.

New Methods

None.

Overriding Methods

somtSetSymbolsOnEntry
somDumpSelfInt

SOMTStringEntryC Class

Description

A **SOMTStringEntryC** object represents a string type definition.

File Stem

scstring

Base

SOMTEntryC

Metaclass

SOMClass

Ancestor Classes

SOMTEntryC, **SOMObject**

Attributes

Listed below is each available attribute, with its corresponding type in parentheses, followed by a description of its purpose:

somtStringLength (long)
The length of the string, as specified in the .idl file. If unspecified, this attribute is zero. This attribute has no “set” method.

New Methods

None.

Overriding Methods

somtSetSymbolsOnEntry
somDumpSelfInt

SOMTStructEntryC Class

Description

A **SOMTStructEntryC** object represents a struct definition or an exception.

Every class entry holds a pointer to a struct entry (**SOMTStructEntryC** object) for each struct defined within the class's interface specification. Each struct entry has attributes that represent the class in which the struct was defined (**somtStructClass**) and whether the struct actually represents an exception (**somtIsException**), and methods for accessing the members of the struct (**somtGetFirstMember** and **somtGetNextMember**). The members of the struct are represented by **SOMTTypedefEntryC** objects whose attributes and methods give the member types and declarator names.

File Stem

scstruct

Base

SOMTEntryC

Metaclass

SOMClass

Ancestor Classes

SOMTEntryC, **SOMObject**

Attributes

Listed below is each available attribute, with its corresponding type in parentheses, followed by a description of its purpose:

somtStructClass (SOMTClassEntryC)
A pointer to an object representing the class in which this struct was defined. This attribute has no "set" method.

somtIsException (boolean)
Whether the struct actually represents an exception. This attribute has no "set" method.

New Methods

somtGetFirstMember
somtGetNextMember

Overriding Methods

somtSetSymbolsOnEntry
somDumpSelfInt

somtGetFirstMember Method

Purpose

The **somtGetFirstMember** method gets the first member for a struct entry.

IDL Syntax

SOMTTypedefEntryC somtGetFirstMember ( );

Note: This method does *not* take an **Environment** parameter.

Description

The **somtGetFirstMember** method returns the first member for the struct entry on which the method was invoked. The next member can be obtained using the corresponding **somtGetNextMember** method.

Note that the **somtGetFirstMember** and **somtGetNextMember** methods cannot be used in doubly nested loops. For example, the following doubly nested loop will not work, because following the first execution of the inner loop, the invocation of **somtGetNextMember** in the outer loop will return NULL:

```
for (m1 = _somtGetFirstMember(struct); m1;
     m1 = _somtGetNextMember(struct))
   for (m2 = _somtGetFirstMember(struct); m2;
        m2 = _somtGetNextMember(struct))
    /* etc. */
```

Parameters

receiver The struct entry whose first member is to be retrieved.

Return Value

This method returns the first member for a struct entry.

Related Information

Methods: somtGetNextMember

somtGetNextMember Method

Purpose

Get the next member for a struct entry, relative to the previous call for a similar entry.

IDL Syntax

SOMTTypedefEntryC somtGetNextMember ( );

Note: This method does *not* take an **Environment** parameter.

Description

The **somtGetNextMember** method returns the next member for the struct entry on which the method was invoked, if it has a next member. Otherwise, it returns NULL.

A call to a **somtGetNextMember** method is relative to the last call of either the same method or the corresponding **somtGetFirstMember** method**,** applied to the same entry object. Note that this implies that the **somtGetFirstMember** and **somtGetNextMember** methods cannot be used in doubly nested loops. For example, the following doubly nested loop will not work, because following the first execution of the inner loop, the invocation of **somtGetNextMember** in the outer loop will return NULL:

```
for (m1 = _somtGetFirstMember(struct); m1;
     m1 = _somtGetNextMember(struct))
   for (m2 = _somtGetFirstMember(struct); m2;
        m2 = _somtGetNextMember(struct))
    /* etc. */
```

Parameters

receiver The struct entry whose next item is to be retrieved.

Return Value

This method returns the next member for the struct entry represented by *receiver*, if it has a next member. Otherwise, it returns NULL.

Related Information

Methods: somtGetFirstMember

SOMTTemplateOutputC Class

Description

An object of class **SOMTTemplateOutputC** represents the output template for an emitter. The template controls the format and content of the sections that an emitter emits. (The emitter itself controls which sections are actually emitted and their order, through its implementation of **somtGenerateSections**.) The template is initialized from the template (.efw) file. The template consists of a set of section names and corresponding text templates containing symbols that, when emitted, are replaced by values appropriate for the emitter's target class/module. For example, the following fragment of a template file specifies the format of the *class* section for a particular emitter:

```
:classS
The class name is <className>.
```

Lines that begin with a colon introduce a section. (By convention, section names end with **S**.) Symbols are contained in angle brackets (e.g., <className>). When the SOM Compiler compiles the definition of class *Foo* and invokes the emitter with the above template, the emitter will produce the following text when it emits the *class* section:

```
The class name is Foo.
```

Lines in a template that begin with "?" are emitted only if at least one symbol appearing on that line is defined with a non-NULL value. In addition to simple symbol substitution, two forms of complex substitution are also supported, list substitution and comment substitution.

Comment substitution is specified by two dashes before the symbol name. For example,

```
<– – methodComment>
```

indicates that the value of the symbol *methodComment* should be emitted in comment form. The comment form used depends on the **somtCommentStyle** and **somtCommentNewline** attributes of the template.

List substitution is specified by "..." preceding the closing angle bracket. In addition, any characters preceding the symbol name indicate the prefix for non-empty lists, and any characters after the symbol name and before the "..." indicate the delimiter for list items. For example,

```
<:methodShortParamNameList, ...>
```

indicates that the items that constitute the value of symbol *methodShortParamNameList* should be emitted with a preceding colon and with a comma and a space between each item, as in

```
:x, y, z
```

Items are delimited in the symbol's value by newline characters. The **somtLineLength** attribute of the template controls how many list items appear on each line.

The **SOMTTemplateOutputC** class provides methods for maintaining the emitter's symbol table. The **SOMTTemplateOutputC** class defines several general-purpose symbols as well as methods through which an emitter can define special-purpose symbols. It also provides methods whereby an emitter can define and emit special-purpose *sections* in addition to those defined by the **SOMTEmitC** class. The general-purpose symbols predefined by the **SOMTemplateOutuptC** class are as follows:

classMajorVersion, classMinorVersion, classSourceFile, classSourceFileStem, classReleaseOrder, classInclude, className, classComment, classLineNumber, classMods, classIDLScopedName, classCScopedName,

baseMajorVersion, baseMinorVersion, baseSourceFile, baseSourceFileStem, baseInclude, baseName, baseIDLScopedName, baseCScopedName, baseComment, baseLineNumber,

metaMajorVersion, metaMinorVersion, metaSourceFile, metaSourceFileStem, metaInclude, metaName, metaIDLScopedName, metaCScopedName, metaComment, metaLineNumber,

dataName, dataIDLScopedName, dataCScopedName, dataComment, dataLineNumber, dataMods, dataType, dataArrayDimensions, dataPointer,

methodName, methodIDLScopedName, methodIDLCScopedName, methodComment, methodLineNumber, methodMods, methodType, methodCReturnType, methodContext, methodRaises, methodClassName, methodCParamList, methodCParamListVA, methodIDLParamList, methodShortParamNameList, methodFullParamNameList,

parameterType, parameterDirection, parameterCDeclaration, parameterIDLDeclaration, parameterName, parameterMods, parameterLineNumber, parameterComment, parameterIDLScopedName, parameterCScopedName,

constantName, constantIDLScopedName, constantCScopedName, constantComment, constantLineNumber, constantMods, constantType, constantValueUnevaluated, constantEvaluated,

typedefComment, typedefLineNumber, typedefBaseType, typedefDeclarators,

structName, structIDLScopedName, structCScopedName, structComment, structLineNumber, structMods,

unionName, unionIDLScopedName, unionCScopedName, unionComment, unionLineNumber, unionMods,

enumName, enumIDLScopedName, enumCScopedName, enumComment, enumLineNumber, enumMods, enumNames,

stringLength, sequenceLength,

attributeDeclarators, attributeBaseType, attributeComment, attributeLineNumber, attributeMods,

passthruName, passthruComment, passthruLineNumber, passthruMods, passthruLanguage, passthruTarget, passthruBody,

moduleName, moduleIDLScopedName, moduleCScopedName, moduleComment, moduleLineNumber, moduleMods, and timeStamp.

The **SOMTTemplateOutputC** class also defines the following symbols, which are used by the **somtEmit**<section> methods (defined by the **SOMTEmitC** class) to determine what section names correspond to different sections. For example, the **somtEmitProlog** method uses the symbol *prologSN* to determine what section name corresponds to the *prolog* section. (The suffix **SN** denotes "section name.") The default value of *prologSN* is "prologS". The remainder of the symbols below follow the same convention:

prologSN, baseIncludesPrologSN, baseIncludesSN, baseIncludesEpilogSN, metaIncludeSN, classSN, metaSN, basePrologSN, baseSN, baseEpilogSN, constantPrologSN, constantSN, constantEpilogSN, typedefPrologSN, typedefSN, typedefEpilogSN, structPrologSN, structSN, structEpilogSN, unionPrologSN, unionSN, unionEpilogSN, enumPrologSN, enumSN, enumEpilogSN, attributePrologSN, attributeSN, attributeEpilogSN, interfacePrologSN, interfaceSN, interfaceEpilogSN, modulePrologSN, moduleSN, moduleEpilogSN, passthruPrologSN, passthruSN, passthruEpilogSN, releaseSN, dataPrologSN, dataSN, dataEpilogSN, methodsPrologSN, methodsSN, overrideMethodsSN, overriddenMethodsSN, inheritedMethodsSN, methodsEpilogSN, epilogSN

File Stem

sctmplt

Base

SOMObject

Metaclass

SOMClass

Ancestor Classes

SOMObject

Types

enum **somtCommentStyleT** {somtDashesE, somtCPPE, somtCSimpleE, somtCBlockE }

Attributes

Listed below is each available attribute, with its corresponding type in parentheses, followed by a description of its purpose:

somtCommentStyle (somtCommentStyleT)
The style in which comments are emitted, as follows:
somtDashesE: "—" at the start of each line
somtCPPE: "//" at the start of each line
somtCSimpleE: simple C style, each line wrapped in "/*" and "*/"
somtCBlockE: block C style, a leading "/*", then a "*" on each line and a final "*/"

somtLineLength (long)
Controls the length of emitted lines, for list output only. The default line length is 72. At least one list item will be output on each line, so making this value very small causes list items to be emitted one per line.

somtCommentNewline (boolean)
If TRUE, each line of comments that are emitted (using the **somtOutputComment** method or by comment substitution in the output template) is preceded by a newline.

New Methods

somtGetSymbol
somtSetSymbol
somtSetSymbolCopyName
somtSetSymbolCopyValue
somtSetSymbolCopyBoth
somtCheckSymbol
somtSetOutputFile
somto
somtOutputComment
somtOutputSection
somtAddSectionDefinitions
somtReadSectionDefinitions
somtExpandSymbol

Overriding Methods

somInit
somUninit
somPrintSelf
somDumpSelfInt

somtAddSectionDefinitions Method

Purpose

Reads section definitions from a string and adds them to a specified template.

IDL Syntax

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

Note: This method does *not* take an **Environment** parameter.

Description

The **somtAddSectionDefinitions** method adds the section definitions specified in *defString* to the template represented by *receiver*. The section definitions in *defString* must be in the following form:

```
:section1
value 1 line 1
value 1 line 2
:section2
value 2 line 1
:section3
value 3 line 1
```

where each line containing "**:**" in column 1 introduces a new section. The section name is the text immediately following the colon. (A backslash in column one can be used to escape a colon that is not used to start a new section.)

Parameters

receiver	An object of class **SOMTTemplateOutputC** representing a template.
defString	A **string** that specifies the section definitions to add to the template.

Return Value

None.

Original Class

SOMTTemplateOutputC

Related Information

Method: somtExpandSymbol, somtReadSectionDefinitions, somtCheckSymbol, somtSetSymbolCopyBoth, somtSetSymbolCopyValue, somtSetSymbolCopyName

somtCheckSymbol Method

Purpose

Checks whether a symbol has been set in a specified template's symbol table.

IDL Syntax

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

Note: This method does *not* take an **Environment** parameter.

Description

The **somtCheckSymbol** method checks whether the specified symbol has a non-NULL, non-zero length value in the template's symbol table.

Parameters

receiver An object of class **SOMTTemplateOutputC** representing a template.

name A **string** representing the name of the symbol to be tested.

Return Value

The **somtCheckSymbol** method returns TRUE if the indicated symbol has a non-NULL, non-zero length value. Otherwise, it returns FALSE.

Example

```
SOMTTemplateOutputC template;
template = __get_somtTemplate(emitter);
if (_somtCheckSymbol(template, "className"))
   printf("The <className> symbol is set.\n");
```

Original Class

SOMTTemplateOutputC

Related Information

Methods: somtExpandSymbol, somtReadSectionDefinitions, somtAddSectionDefinitions, somtSetSymbolCopyBoth, somtSetSymbolCopyValue, somtSetSymbolCopyName, somtGetSymbol

somtExpandSymbol Method

Purpose

Expands a section from a template file.

IDL Syntax

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

Note: This method does *not* take an **Environment** parameter.

Description

The **somtExpandSymbol** method expands a section from a template file, given a symbol representing the name of the section, by substituting symbol values for symbol names in the template. This expansion can then be assigned as the value of another symbol, using one of the **somtSetSymbol...** methods. In this way, the values of emitter symbols can be defined declaratively in the template file, rather than procedurally within the emitter's code.

Parameters

receiver	An object of class **SOMTTemplateOutputC** representing a template.
s	A **string** representing the name of the section to be expanded.
buf	A **string** representing the buffer which will receive the expanded section.

Return Value

The **somtExpandSymbol** method expands the specified section in *buf* and returns *buf*.

Example

If the template (.efw) file for an emitter contains the following section definition:

```
:methodPrefixS
<functionprefix>_
```

then the following code within an overriding implementation of the **somtGenerateSections** method (or any other method of **SOMTEmitC**) will set symbol "methodPrefix" to be the expansion of the methodPrefixS section in the template file (that is, the value of symbol "functionprefix," if defined by the emitter, followed by an underscore).

```
SOMTTemplateOutputC template = __get_somtTemplate(emitter);
char buf[MAX_SYMBOL_SIZE];
...
_somtSetSymbolCopyBoth(template, "methodPrefix",
                _somtExpandSymbol(template, "methodPrefixS", buf))
```

Original Class

SOMTTemplateOutputC

Related Information

Methods: somtReadSectionDefinitions, somtAddSectionDefinitions, somtCheckSymbol, somtSetSymbolCopyBoth, somtSetSymbolCopyValue, somtSetSymbolCopyName, somtSetSymbol, somtGetSymbol

somtGetSymbol Method

Purpose

Gets a symbol value from a template's symbol table.

IDL Syntax

string somtGetSymbol (
in string *name***);**

Note: This method does *not* take an **Environment** parameter.

Description

The **somtGetSymbol** method gets the value of symbol *name* from the symbol table of the template object on which the method was invoked.

Parameters

receiver An object of class **SOMTTemplateOutputC** representing a template.

name A **string** representing the name of the symbol whose value is needed.

Return Value

The **somtGetSymbol** method returns the string representing the value of the symbol. If there is no associated value, then **somtGetSymbol** returns NULL.

Example

To set the symbol **prefix** to the value of **externalPrefix** (if that symbol has already been given a value):

```
SOMTTemplateOutputC template = __get_somtTemplate(emitter);
_somtSetSymbolCopyName(template, "prefix",
             _somtGetSymbol(template, "externalPrefix"));
```

Original Class

SOMTTemplateOutputC

Related Information

Methods: somtExpandSymbol, somtCheckSymbol, somtSetSymbolCopyBoth, somtSetSymbolCopyValue, somtSetSymbolCopyName

somto Method

Purpose

Outputs a template to a file.

IDL Syntax

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

Note: This method does *not* take an **Environment** parameter.

Description

The **somto** method outputs a template *tmplt* after substituting for any symbols that occur in it. (This method is usually not called directly.) Five kinds of symbol substitutions are supported: **simple**, **list**, **comment**, **tab**, and **conditional**.

Substitutable items in the template are bracketed with angle brackets (**< >**). The backslash (****) can be used to escape an angle bracket.

Simple substitution replaces a symbol with its value. If the symbol has no value in the symbol table of the **SOMTTemplateOutputC** object on which the method was invoked, then the symbol is replaced by the string "Symbol <...> is not defined."

List substitution replaces a symbol with a value expressed in list form, using specified delimiters. The symbol value must consist of a sequence of list items, separated by newline characters. The list-substitution specification consists of four parts: a **prefix**, the symbol**,** a **separator**, and a **list indicator**. The **prefix** and **separator** components can only be composed of blanks, commas, colons, and semicolons. 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 precede the list whenever there is at least one item in the list, and the separator will be used between any two list items. After each item of the list is output, the next item is evaluated to determine whether it would exceed the maximum line length (set by the *receiver*'s attribute **somtLineLength**). If it would, then a new line is begun and the next value is placed directly under the first item.

Comment substitution replaces a symbol with its value in the form of a comment. A comment specification consists of the two characters "– –" followed by a symbol name. For example, `<- - classComment>` is a valid comment-substitution specification. The lines of the comment are output according to the **somtCommentStyle** attribute of the *receiver*, and are aligned with the starting column of the comment specification.

Tab substitution is specified by <@*dd*>, where *dd* is a valid positive integer representing a column number. 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 unless at least one valid, non-blank symbol substitution occurs on the line.

Parameters

receiver — An object of class **SOMTTemplateOutputC** representing an emitter's template.

tmplt — A **string** representing the template to be output.

Return Value

The **somto** method has no return value. The template is output to the file associated with the **SOMTTemplateOutputC** object.

somtOutputComment Method

Purpose

Inserts a comment into the output file.

IDL Syntax

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

Note: This method does *not* take an **Environment** parameter.

Description

The **somtOutputComment** method inserts a comment into the output file. The specified *comment* must be a string in which each comment line terminates with a newline character.

Parameters

receiver	An object of class **SOMTTemplateOutputC** representing a template.
comment	A **string** representing the comment to be emitted.

Return Value

None.

Example

```
SOMTTemplateOutputC template = __get_somtTemplate(emitter);
_somtOutputComment(template, "Here is a comment to emit.");
```

Original Class

SOMTTemplateOutputC

Related Information

Methods: somtOutputSection, somto, somtSetOutputFile

somtOutputSection Method

Purpose

Outputs a section of a template.

IDL Syntax

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

Note: This method does *not* take an **Environment** parameter.

Description

The **somtOutputSection** method outputs the section named by *sectionName* after substituting for any symbols that occur in that section. [The template (.efw) file defines each section.] Five types of symbol substitution are supported: **simple**, **list**, **comment**, **tab**, and **conditional**.

Substitutable items in the template are bracketed with angle brackets (**< >**). The backslash (****) can be used to escape an angle bracket.

Simple substitution replaces a symbol with its value. If the symbol has no value for the specified **SOMTTemplateOutputC** object, then the symbol is replaced by the string "Symbol <...> is not defined."

List substitution replaces a symbol with a value expressed in list form, using specified delimiters. The symbol value must consist of a sequence of list items, separated by newline characters. The list-substitution specification consists of four parts: a **prefix**, the symbol**,** a **separator**, and a **list indicator**. The **prefix** and **separator** components can only be composed of blanks, commas, colons, and semicolons. 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 precede the list whenever there is at least one item in the list, and the separator will be used between any two list items. After each item of the list is output, the next item is evaluated to determine whether it would exceed the maximum line length (set by the *receiver*'s attribute **somtLineLength**). If it would, then a new line is begun and the next value is placed directly under the first item.

Comment substitution replaces a symbol with its value in the form of a comment. A comment specification consists of the two characters "– –" followed by a symbol name. For example, `<- - classComment>` is a valid comment-substitution specification. The lines of the comment are output according to the **somtCommentStyle** attribute of the *receiver*, and are aligned with the starting column of the comment specification.

Tab substitution is specified by <@*dd*>, where *dd* is a valid positive integer representing a column number. 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 unless at least one valid, non-blank symbol substitution occurs on the line.

Parameters

receiver An object of class **SOMTTemplateOutputC** representing a template.

sectionName A **string** representing the name of the section to be emitted.

Return Value

The **somtOutputSection** method has no return value. The section is output to the file associated with the **SOMTTemplateOutputC** object.

Example

```
SOMTTemplateOutputC template = __get_somtTemplate(emitter);
_somtOutputSection(template, ”metaSectionS”);
```

Original Class

SOMTTemplateOutputC

Related Information

Methods: somtReadSectionDefinitions, somtAddSectionDefinitions, somto, somtSetOutputFile

somtReadSectionDefinitions Method

Purpose

Reads section definitions from a file and adds them to the specified template.

IDL Syntax

void somtReadSectionDefinitions (inout FILE *_fp_);

Note: This method does *not* take an **Environment** parameter.

Description

The **somtReadSectionDefinitions** method reads all section definitions from the file specified by *fp* and adds them to the template on which the method was invoked. Section definitions must be in the following form:

```
:section1
value 1 line 1
value 1 line 2
:section2
value 2 line 1
:section3
value 3 line 1
```

where each line containing "**:**" in column 1 introduces a new section. The section name is the text immediately following the colon. (A backslash in column one can be used to escape a colon that is not used to start a new section.)

Parameter

receiver — An object of class **SOMTTemplateOutputC** representing the template to which the section definitions will be added.

fp — A pointer to the file containing the section definitions.

Return Value

None.

Example

To read the section definitions from the **myfile.efw** template file:

```
MyEmitter emitter;
SOMTTemplateOutputC template;

emitter = MyEmitterNew();
template = __get_somtTemplate(emitter);

if (deffile = _somtOpenSymbolsFile(emitter, "myfile.efw", "r"))
    _somtReadSectionDefinitions(template, deffile);
```

Original Class

SOMTTemplateOutputC

Related Information

Methods: somtAddSectionDefinitions, somtOutputSection, somto, somtSetOutputFile, somtOpenSymbolsFile

somtSetOutputFile Method

Purpose

Sets the output file for an emitter.

IDL Syntax

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

Note: This method does *not* take an **Environment** parameter.

Description

The **somtSetOutputFile** method specifies the file to which all output will be directed. This method usually need not be invoked directly, because a template's output file is set when its emitter's target file is set (using **_set_somtTargetFile**).

The default output file is **stdout**.

Parameters

receiver	An object of class **SOMTTemplateOutputC** representing the template of the emitter.
fp	A pointer to the output file.

Return Value

None.

Original Class

SOMTTemplateOutputC

Related Information

Methods: somtOutputSection, somto

somtSetSymbol Method

Purpose

Sets a symbol to a given value in the symbol table of a specified template.

IDL Syntax

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

Note: This method does *not* take an **Environment** parameter.

Description

The **somtSetSymbol** method sets a symbol *name* to a specified *value* in the symbol table of the template object on which the method was invoked. This adds the *name/value* pair to the symbol table or overwrites a previous setting, if necessary.

The symbol table assumes ownership of both the *name* and *value*, and these strings must not be freed by the caller. If the *value* of the named symbol is changed by subsequent calls to a **somtSetSymbol...** method, then the string passed as the *value* parameter will be freed by the symbol table. Hence, if the string representing the *value* is a static string or a string that will be freed by the caller, or if the symbol value may change during subsequent execution of the emitter and it is necessary that the string passed as the *value* parameter not be freed by the symbol table, then you should use the **somtSetSymbolCopyValue** or **somtSetSymbolCopyBoth** method to define the *name/value* pair. Likewise, if the string representing the *name* is a static string or a string that will be freed by the caller, you should use the **somtSetSymbolCopyName** or **somtSetSymbolCopyBoth** method to define the *name/value* pair.

Parameters

receiver	An object of class **SOMTTemplateOutputC**, representing the template object of an emitter.
name	A **string** representing a symbol name.
value	A **string** representing a symbol value.

Return Value

None.

Original Class

SOMTTemplateOutputC

Related Information

Methods: somtExpandSymbol, somtCheckSymbol, somtSetSymbolCopyBoth, somtSetSymbolCopyValue, somtSetSymbolCopyName, somtGetSymbol, somtSetSymbol

somtSetSymbolCopyBoth Method

Purpose

Sets a symbol to a given value in the symbol table of a specified template, using copies of the original name and value.

IDL Syntax

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

Note: This method does *not* take an **Environment** parameter.

Description

The **somtSetSymbolCopyBoth** method sets a symbol *name* to a specified *value* in the symbol table of the template object on which the method is invoked. This adds the *name/value* pair to the symbol table or overwrites a previous setting, if necessary.

The **somtSetSymbolCopyBoth** method makes a copy of both the *name* and *value* parameters; it stores in the symbol table (and takes ownership of) the copies, rather than the original strings. This method is appropriate when the caller wants to maintain ownership of the strings representing both *name* and *value*, or when the *name* and *value* are static strings. Because the method makes a copy of *value* before storing it in the symbol table, if the value of the symbol is subsequently changed, only the symbol table's copy of the original *value* will be freed, and not the string passed by the caller.

Parameters

receiver	An object of class **SOMTTemplateOutputC** representing the template object of an emitter.
name	A **string** representing a symbol name.
value	A **string** representing a symbol value.

Return Value

None.

Example

To set the symbol `newMethodLabel` to the value `"New Sections"`:

```
SOMTTemplateOutputC t = __get_somtTemplate(emitter);
_somtSetSymbolCopyBoth(t,"newMethodLabel", "New Sections");
```

Original Class

SOMTTemplateOutputC

Related Information

Methods: somtExpandSymbol, somtCheckSymbol, somtSetSymbolCopyValue, somtSetSymbolCopyName, somtGetSymbol, somtSetSymbol

somtSetSymbolCopyName Method

Purpose

Sets a symbol to a given value in the symbol table of a specified template, using a copy of the original name.

IDL Syntax

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

Note: This method does *not* take an **Environment** parameter.

Description

The **somtSetSymbolCopyName** method sets a symbol *name* to a specified *value* in the symbol table of the template object on which the method is invoked. This adds the *name/value* pair to the symbol table or overwrites a previous setting, if necessary.

The **somtSetSymbolCopyName** method makes a copy of the *name* parameter, but not the *value* parameter, before storing the *name/value* pair in the symbol table. This method is appropriate when the caller wants to maintain ownership of the string representing the symbol *name* or when the *name* is a static string.

After execution of the **somtSetSymbolCopyName** method, the symbol table assumes ownership of the string passed as the *value* parameter. Hence, this string must not be freed by the caller, and it should not be a static string. If the *value* of the named symbol is changed by subsequent calls to a **somtSetSymbol...** method, then the string passed as the *value* parameter will be freed by the symbol table. If the string representing the *value* is a static string or a string that will be freed by the caller, or if the symbol *value* may change during subsequent execution of the emitter and it is necessary that the string passed as the *value* parameter not be freed by the symbol table, then you should use the **somtSetSymbolCopyBoth** method to define the *name/value* pair.

Parameters

receiver	An object of class **SOMTTemplateOutputC** representing the template object of an emitter.
name	A **string** representing a symbol name.
value	A **string** representing a symbol value.

Return Value

None.

Example

To set the `"baseNames"` variable to the value returned by a function *buildbaseNames* that returns ownership of the string it produces to the caller:

```
SOMTClassEntryC cls =  __get_somtTargetClass(emitter);
SOMTTemplateOutputC t = __get_somtTemplate(emitter);

_somtSetSymbolCopyName(t,"baseNames", buildbaseNames(cls));
```

Original Class

SOMTTemplateOutputC

Related Information

Method: somtExpandSymbol, somtCheckSymbol, somtSetSymbolCopyBoth, somtSetSymbolCopyValue, somtGetSymbol, somtSetSymbol

somtSetSymbolCopyValue Method

Purpose

Sets a symbol to a given value in the symbol table of a template object, using a copy of the original value.

IDL Syntax

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

Note: This method does *not* take an **Environment** parameter.

Description

The **somtSetSymbolCopyValue** method sets a symbol *name* to a specified *value* in the symbol table of the template object on which the method is invoked. This adds the *name/value* pair to the symbol table or overwrites a previous setting, if necessary.

The **somtSetSymbolCopyValue** method makes a copy of the *value* parameter, but not the *name* parameter, before storing the *name/value* pair in the symbol table. This method is appropriate when the caller wants to maintain ownership of the string representing the symbol *value* or when the *value* is a static string. Because the method makes a copy of the *value* before storing it in the symbol table, if the *value* of the symbol is subsequently changed, only the symbol table's copy of the original *value* will be freed, and not the string passed by the caller.

After execution of this method, the symbol table assumes ownership of the string passed as the *name* parameter. Hence, this string must not be freed by the caller, and it should not be a static string.

Parameters

receiver	An object of class **SOMTTemplateOutputC** representing the template object of an emitter.
name	A **string** representing the symbol name.
value	A **string** representing the symbol value.

Return Value

None.

Example

To change the default value of the"className" symbol so that it begins with an underscore:

```
char *s;
SOMTTemplateOutputC t = __get_somtTemplate(emitter);
char buf[MAX_SMALL_STRING];
...
s = _somtGetSymbol(t, "className");
if (s && *s)
    {
       sprintf(buf, "_%s", s);
       _somtSetSymbolCopyValue(t, "className", buf);
    }
```

Original Class

SOMTTemplateOutputC

Related Information

Methods: somtExpandSymbol, somtCheckSymbol, somtSetSymbolCopyBoth, somtSetSymbolCopyName, somtGetSymbol, somtSetSymbol

SOMTTypedefEntryC Class

Description

A **SOMTTypedefEntryC** object represents a typedef within a class definition or a member of a user-defined struct. Each typedef entry has an attribute representing the base type (**somtTypedefType**) of the new type(s) and methods for accessing the declarator names (**somtGetFirstDeclarator** and **somtGetNextDeclarator**) of the typedef. If the type of a typedef is a user-defined type, then the **somtTypedefType** attribute will be a pointer to an instance of **SOMTUserDefinedTypeEntryC**. The **somtOriginalTypedef** attribute of that object will point to a **SOMTTypedefEntryC** object that represents the typedef for that user-defined type.

For example, if the following appears in an IDL specification:

```
typedef long mytype1;
typedef mytype1 mytype2;
```

the **SOMTTypedefEntryC** object that represents the typedef of “mytype2” would have a **somtTypedefType** attribute whose value is an object of type **SOMTUserDefinedTypeEntryC**. That object’s **somtOriginalTypedef** attribute would point to a **somtTypedefEntryC** object that represents the typedef of “mytype1.”

Because a single typedef may have several declarators (that introduce several user-defined types), the **somtTypedefType** attribute of a typedef gives only the *base* type of the user-defined types; to get the full type, users should access each declarator in turn and get its **somtType** attribute.

File Stem

sctdef

Base

SOMTEntryC

Metaclass

SOMClass

Ancestor Classes

SOMTEntryC, **SOMObject**

Attributes

Listed below is each available attribute, with its corresponding type in parentheses, followed by a description of its purpose:

somtTypedefType (SOMTEntryC)
A pointer to an entry object representing the base type of the typedef. This doesn’t include pointer stars or array declarators; to get the full type, get each of the declarators (using **somtGetFirstDeclarator** and **somtGetNextDeclarator**) and get its **somtType** attribute. This attribute has no “set” method.

New Methods

somtGetFirstDeclarator
somtGetNextDeclarator

Overriding Methods

somtSetSymbolsOnEntry
somDumpSelfInt

somtGetFirstDeclarator Method

Purpose

The **somtGetFirstDeclarator** method gets the first declarator for a typedef entry.

IDL Syntax

SOMTCommonEntryC somtGetFirstDeclarator ( );

Note: This method does *not* take an **Environment** parameter.

Description

The **somtGetFirstDeclarator** method returns the first declarator for the typedef entry on which the method was invoked. The next declarator can be obtained using the corresponding **somtGetNextDeclarator** method.

Note that the **somtGetFirstDeclarator** and **somtGetNextDeclarator** methods cannot be used in doubly nested loops. For example, the following doubly nested loop will not work, because following the first execution of the inner loop, the invocation of **somtGetNextDeclarator** in the outer loop will return NULL:

```
for (d1 = _somtGetFirstDeclarator(myTypedef); d1;
     d1 = _somtGetNextDeclarator(myTypedef))
   for (d2 = _somtGetFirstDeclarator(myTypedef); d2;
        d2 = _somtGetNextDeclarator(myTypedef))
    /* etc. */
```

Parameters

receiver The typedef entry whose first declarator is to be retrieved.

Return Value

This method returns the first declarator for a typedef entry.

Example

To iterate through the declarators of a typedef:

```
SOMTCommonEntryC myEntry;

printf("List of declarators:\n");
for (myEntry = _somtGetFirstDeclarator(myTypedef); myEntry;
    myEntry = _somtGetNextDeclarator(myTypedef))
    printf("%s\n", __get_somtEntryName(myEntry));
```

Related Information

Methods: somtGetNextDeclarator

somtGetNextDeclarator Method

Purpose

This method gets the next declarator for a typedef entry, relative to the previous call for a similar entry.

IDL Syntax

SOMTCommonEntryC somtGetNextDeclarator ( );

Note: This method does *not* take an **Environment** parameter.

Description

The **somtGetNextDeclarator** method returns the next declarator for the typedef entry on which the method was invoked, if it has a next declarator. Otherwise, it returns NULL.

A call to a **somtGetNextDeclarator** method is relative to the last call of either the same method or the corresponding **somtGetFirstDeclarator** method**,** applied to the same entry object. Note that this implies that the **somtGetFirstDeclarator** and **somtGetNextDeclarator** methods cannot be used in doubly nested loops. For example, the following doubly nested loop will not work, because following the first execution of the inner loop, the invocation of **somtGetNextDeclarator** in the outer loop will return NULL:

```
for (d1 = _somtGetFirstDeclarator(myTypedef); d1;
     d1 = _somtGetNextDeclarator(myTypedef))
   for (d2 = _somtGetFirstDeclarator(myTypedef); d2;
        d2 = _somtGetNextDeclarator(myTypedef))
    /* etc. */
```

Parameters

receiver The entry whose next item is to be retrieved.

Return Value

This method returns the next declarator for the typedef entry represented by *receiver*, if it has a next item of that type. Otherwise, it returns NULL.

Example

To iterate through the declarators of a typedef:

```
SOMTCommonEntryC myEntry;

printf("List of declarators:\n");
for (myEntry = _somtGetFirstDeclarator(myTypedef); myEntry;
    myEntry = _somtGetNextDeclarator(myTypedef))
    printf("%s\n", __get_somtEntryName(myEntry));
```

Related Information

Methods: somtGetFirstDeclarator

SOMTUnionEntryC Class

Description

A **SOMTUnionEntryC** object represents a union definition. It provides attributes and methods for accessing the union's switch type and each of its cases.

File Stem

scunion

Base

SOMTEntryC

Metaclass

SOMClass

Ancestor Classes

SOMTEntryC, **SOMObject**

Types

```
struct somtLabelList {
    string label;
    somtLabelList *nextLabel;
};

struct somtCaseEntry {
    somtLabelList *caseLabels;
    SOMTEntryC    memberType;
    SOMTDataEntryC   memberDeclarator;
};
```

Attributes

Listed below is each available attribute, with its corresponding type in parentheses, followed by a description of its purpose:

somtSwitchType (SOMTEntryC)
A pointer to an entry object representing the switch type of the union.
This attribute has no "set" method.

New Methods

somtGetFirstCaseEntry
somtGetNextCaseEntry

Overriding Methods

somtSetSymbolsOnEntry
somDumpSelfInt

somtGetFirstCaseEntry Method

Purpose

The **somtGetFirstCaseEntry** method gets the first case for a union entry.

IDL Syntax

somtCaseEntry * somtGetFirstCaseEntry ( );

Note: This method does *not* take an **Environment** parameter.

Description

The **somtGetFirstCaseEntry** method returns the first case for the union entry on which the method was invoked. The next case can be obtained using the corresponding **somtGetNextCaseEntry** method.

Note that the **somtGetFirstCaseEntry** and **somtGetNextCaseEntry** methods cannot be used in doubly nested loops. For example, the following doubly nested loop will not work, because following the first execution of the inner loop, the invocation of **somtGetNextCaseEntry** in the outer loop will return NULL:

```
for (c1 = _somtGetFirstCaseEntry(myUnion); c1;
     c1 = _somtGetNextCaseEntry(myUnion))
   for (c2 = _somtGetFirstCaseEntry(myUnion); c2;
        c2 = _somtGetNextCaseEntry(myUnion))
    /* etc. */
```

Parameters

receiver The union entry whose first case is to be retrieved.

Return Value

This method returns the first case for a union entry. The **somtGetFirstCaseEntry** method returns a pointer to a **somtCaseEntry** struct; see the reference page for **SOMTUnionEntryC** class for the definition of **somtCaseEntry**.

Example

To iterate through the cases of a union:

```
SOMTCaseEntry *case;

printf("List of cases:\n");
for (case = _somtGetFirstCaseEntry(myUnion); case;
     case = _somtGetNextCaseEntry(myUnion))
   printf("%s\n", __get_somtEntryName(case->memberDeclarator));
```

Related Information

Methods: somtGetNextCaseEntry

somtGetNextCaseEntry Method

Purpose

The **somtGetNextCaseEntry** method gets the next case for a union entry.

IDL Syntax

somtCaseEntry * somtGetNextCaseEntry ( );

Note: This method does *not* take an **Environment** parameter.

Description

The **somtGetNextCaseEntry** method returns the next case for the union entry on which the method was invoked, if it has a next case. Otherwise, it returns NULL.

A call to a **somtGetNextCaseEntry** method is relative to the last call of either the same method or the corresponding **somtGetFirstCaseEntry** method**,** applied to the same entry object. Note that this implies that the **somtGetFirstCaseEntry** and **somtGetNextCaseEntry** methods cannot be used in doubly nested loops. For example, the following doubly nested loop will not work, because following the first execution of the inner loop, the invocation of **somtGetNextCaseEntry** in the outer loop will return NULL:

```
for (c1 = _somtGetFirstCaseEntry(myUnion); c1;
     c1 = _somtGetNextCaseEntry(myUnion))
   for (c2 = _somtGetFirstCaseEntry(myUnion); c2;
        c2 = _somtGetNextCaseEntry(myUnion))
    /* etc. */
```

Parameters

receiver The union entry whose next case is to be retrieved.

Return Value

This method returns the next case for a union entry. The **somtGetNextCaseEntry** method returns a pointer to a **somtCaseEntry** struct; see the reference page for **SOMTUnionEntryC** class for the definition of **somtCaseEntry**.

Example

To iterate through the cases of a union:

```
SOMTCaseEntry *case;

printf("List of cases:\n");
for (case = _somtGetFirstCaseEntry(myUnion); case;
     case = _somtGetNextCaseEntry(myUnion))
   printf("%s\n", __get_somtEntryName(case->memberDeclarator));
```

Related Information

Methods: somtGetFirstCaseEntry

SOMTUserDefinedTypeEntryC Class

Description

A **SOMTUserDefinedTypeEntryC** object represents a type defined via a “typedef” statement in a .idl file.

File Stem

scusrtyp

Base

SOMTEntryC

Metaclass

SOMClass

Ancestor Classes

SOMTEntryC, **SOMObject**

Attributes

Listed below is each available attribute, with its corresponding type in parentheses, followed by a description of its purpose:

somtOriginalTypedef (SOMTTypedefEntryC)
A pointer to the object representing the typedef that defines the user-defined type.

somtBaseTypeObj (SOMTEntryC)
A pointer to the object representing the base type (short, float, unsigned long, etc.) of the user-defined type, regardless of any intermediate user-defined types.
For example, given

```
typedef short x;
typedef x y;
```

The base type of user-defined type “y” is “short.”

New Methods

None.

Overriding Methods

somDumpSelfInt
somtSetSymbolsOnEntry,
_get_somtTypeObj

Reference for Emitter Framework Functions

somterror Function

Purpose

Prints an error message and increments the error count maintained by the SOM Compiler.

Syntax

void somterror (string *file,* **long** *lineno,* **string** *format, ...);*

Description

The **somterror** function prints an error message and increments the error count maintained by the SOM Compiler. The error message begins with the string “error: ” and includes the name of the file and the line number on which the error occurred, if specified.

Parameters

file	The name of the file in which the error occurred, or NULL.
lineno	The line number on which the error occurred, or zero.
format	A format string suitable for passing to the **printf** C library function.
varargs	The arguments to be passed to **printf.**

Return Value

None.

Example

```
somterror(__get_somtSourceFileName(cls),
          __get_somtSourceLineNumber(entry),
          ”I don’t understand the entry named %s.\n”,
          __get_somtEntryName(entry));
```

Related Information

Functions: somtmsg, somtwarn, somtfatal, somtinternal

somtfatal Function

Purpose

Prints a fatal error message and increments the internal error count maintained by the SOM Compiler.

Syntax

void somtfatal (string *file,* **long** *lineno,* **string** *format, ...);*

Description

The **somtfatal** function prints a fatal error message and increments the internal error count maintained by the SOM Compiler. The error message begins with the string “fatal error: ” and includes the name of the file and the line number on which the error occurred, if specified. After printing the error message, the routine removes the output file and terminates the process.

Parameters

file	The name of the file in which the error occurred, or NULL.
lineno	The line number on which the error occurred, or zero.
format	A format string suitable for passing to the **printf** C library function.
varargs	The arguments to be passed to **printf.**

Return Value

None.

Example

```
somtfatal(__get_somtSourceFileName(cls),
          __get_somtSourceLineNumber(entry),
          ”The entry named %s is a disaster!.\n”,
          __get_somtEntryName(entry));
```

Related Information

Functions: somtmsg, somtwarn, somterror, somtinternal

somtfclose Function

Purpose

Closes a file opened using **somtOpenEmitFile.**

Syntax

int somtfclose (FILE *_fp);_

Description

The **somtfclose** function closes a file opened using **somtOpenEmitFile.** Emitters that use **somtOpenEmitFile** should use this function, rather than **fclose**, to close the output file so that, regardless of the way the standard C library is packaged or whether emitters are statically or dynamically loaded, files opened with **somtOpenEmitFile** will be properly closed. Emitters are not *required* to close their output files; normally, an emitter's return value is the file handle for the file it opened using **somtOpenEmitFile**. If an emitter needs to close its output file, however, the **somtfclose** function should be used, rather than **fclose**.

Parameters

fp A pointer to the file to be closed.

Return Value

The **somtfclose** returns the same return code as the C library **fclose** function.

Example

```
FILE *fp = somtopenEmitFile("hello.foo", "foo");
__set_somtTargetFile(emitter, fp);
...
somtfclose(fp);
```

Related Information

Functions: somtOpenEmitFile

somtGetObjectWrapper Function

Purpose

Gets the entry object corresponding to the *cls* argument passed by the SOM Compiler to an emitter's driver program. This object should then be set as the target class or module of the emitter.

Syntax

**SOMTEntryC somtGetObjectWrapper (Entry ****entry);*

Description

The **somtGetObjectWrapper** function gets the entry object corresponding to the *cls* argument passed by the SOM Compiler to an emitter's driver program. This object should then be set as the target class or module of the emitter. Before freeing the emitter object, the object returned by this function should be freed.

Parameters

entry The data structure passed by the SOM Compiler to an emitter's driver program.

Return Value

The **somtGetObjectWrapper** function returns the entry object created.

Example

```
SOMTClassEntryC oCls;
SOMTModuleEntryC mod;
MyEmitter emitter;
if (cls->type == SOMTClassE) {
    oCls = (SOMTClassEntryC) somtGetObjectWrapper(cls);
    emitter = MyEmitterNew();
    __set_somtTargetClass(emitter, oCls);
...
}
else if (cls->type == SOMTModuleE) {
    mod = (SOMTModuleEntryC) somtGetObjectWrapper(cls);
    emitter = MyEmitterNew();
    __set_somtTargetModule(emitter, mod);
...
}
```

somtinternal Function

Purpose

Prints an internal error message and increments the internal error count maintained by the SOM Compiler.

Syntax

void somtinternal (string *file,* **long** *lineno,* **string** *format, ...);*

Description

The **somtinternal** function prints an internal error message and increments the internal error count maintained by the SOM Compiler. The error message begins with the string “internal error: ” and includes the name of the file and the line number on which the error occurred, if specified. After printing the error message, the routine removes the output file and terminates the process.

Parameters

file	The name of the file in which the error occurred, or NULL.
lineno	The line number on which the error occurred, or zero.
format	A format string suitable for passing to the **printf** C library function.
varargs	The arguments to be passed to **printf.**

Return Value

None.

Example

```
somtinternal(__get_somtSourceFileName(cls),
             __get_somtSourceLineNumber(entry),
             ”I really messed this one up!\n”);
```

Related Information

Functions: somtmsg, somtwarn, somtfatal, somterror

somtmsg Function

Purpose

Prints an informational message.

Syntax

void somtmsg (string *file,* **long** *lineno,* **string** *format, ...);*

Description

The **somtmsg** function prints an informational message. The message includes the name of the file and the line number to which the message applies, if specified.

In order for a **somtmsg** function to produce output, you also must specify the **–v** (verbose) flag when entering the **sc** command on the command line to invoke the SOM Compiler. (See also "Running the SOM Compiler" in chapter 4, "SOM IDL and the SOM Compiler" in the *SOMobjects Developer Toolkit Users Guide*.)

Parameters

file	The name of the file to which the message applies, or NULL.
lineno	The line number to which the message applies, or zero.
format	A format string suitable for passing to the **printf** C library function.
varargs	The arguments to be passed to **printf.**

Return Value

None.

Example

```
somtmsg(__get_somtSourceFileName(cls),
           __get_somtSourceLineNumber(entry),
           "I really like the entry named %s.\n",
           __get_somtEntryName(entry));
```

Related Information

Functions: somterror, somtwarn, somtfatal, somtinternal

somtNewSymbol Function

Purpose

Creates a new symbol name by concatenating a prefix and a name.

Syntax

string somtNewSymbol (string *prefix,* **string** *stem***);**

Description

The **somtNewSymbol** function creates a new symbol name by concatenating a prefix and a name. Ownership of the string is passed to the caller. Hence, the **somtSetSymbol** or **somtSetSymbolCopyValue** method should be used to give the resulting symbol a value (instead of **somtSetSymbolCopyName** or **somtSetSymbolCopyBoth**). This function is useful for overriding implementations of the **somtSetSymbolsOnEntry** method, which takes the prefix as an argument.

Parameters

prefix The prefix of the symbol to be created.

stem The base name of the symbol to be created.

Return Value

The **somtNewSymbol** function returns the new symbol name. Ownership of the string is passed to the caller.

Example

The following code creates the new symbol name “classComment” and sets its value:

```
SOMTClassEntryC class = __get_somtTargetClass(emitter);
SOMTTemplateOutputC template = __get_somtTemplate(emitter);
string comment = __get_somtEntryComment(class);
_somtSetSymbolCopyValue(template,
                        somtNewSymbol(”class”, ”Comment”),
                        (comment ? comment : ””));
```

Related Information

Methods: somtSetSymbol, somtSetSymbolCopyName, somtSetSymbolCopyValue, somtSetSymbolCopyBoth

somtopenEmitFile Function

Purpose

Open an output file for an emitter.

Syntax

FILE * somtopenEmitFile (char **file*, char **ext*)

Description

The **somtopenEmitFile** function opens the named output file for an emitter. It also sets global variables needed by other library functions and adds a header to the newly opened file if *ext* is a known extension. Users can extend the list of known extensions by adding them to the value of the SMKNOWNEXTS environment variable, separated by a semicolon.

Depending on the setting of a global variable (set by the SOM Compiler), the file will be opened for either writing or appending. When an emitter is invoked for the first time on a particular .idl file, this global variable indicates that the file should be opened for writing. When the same emitter is invoked subsequently on the same input file (for example, to process interface definitions within a module definition), the global variable indicates that the file should be opened for appending. In this way, all output for a single input file goes to the same output file, even though the emitter may be invoked multiple times.

When an interrupt occurs as an emitter is executing, the file opened by **somtopenEmitFile** is removed. If you wish to prevent this during critical portions of the code, call the function **somtunsetEmitSignals** at the beginning of your code segment, and call the function **somtresetEmitSignals** after the segment.

Parameters

file — The name of the file to be opened. If NULL, *stdout* is returned.

ext — The extension of the file to be opened. If the specified filename does not include this extension (or if it includes a different extension), a filename is constructed that has the specified extension.

Return Value

The **somtopenEmitFile** functions returns a pointer to the opened file (or *stdout*, if no filename is specified).

Example

```
FILE *fp = somtopenEmitFile("hello.foo", "foo");
__set_somtTargetFile(emitter, fp);
...
somtfclose(fp);
```

Related Information

Functions: somtunsetEmitSignals, somtresetEmitSignals

somtresetEmitSignals Function

Purpose

Resumes signal processing after disabling it via the **somtunsetEmitSignals** function.

Syntax

void somtresetEmitSignals ( );

Description

The **somtresetEmitSignals** function resumes signal processing after disabling it via the **somtunsetEmitSignals** function.

Parameters

None.

Return Value

None.

Example

```
somtunsetEmitSignals();
/* do some protected processing */
somtresetEmitSignals();
```

Related Information

Functions: somtunsetEmitSignals

somtunsetEmitSignals Function

Purpose

Prevents signals from being received as an emitter is executing.

Syntax

void somtunsetEmitSignals ( );

Description

The **somtunsetEmitSignals** function prevents signals from being received as an emitter is executing. Normally, signals such as internal errors and user-generated interrupts are trapped within emitters. It may be necessary, however, to prevent such interrupts from occurring in certain sections of an emitter's code.

This function is also useful for preventing the output file from being removed if an interrupt occurs. Normally, when an interrupt occurs, the file opened by **somtopenEmitFile** is removed. If you wish to prevent this during critical portions of the code, call **somtunsetEmitSignals** at the beginning of your code segment, and call **somtresetEmitSignals** after the segment.

Parameters

None.

Return Value

None.

Example

```
somtunsetEmitSignals();
/* do some protected processing */
somtresetEmitSignals();
```

Related Information

Functions: somtresetEmitSignals, somtOpenEmitFile

somtwarn Function

Purpose

Prints a warning message and increments the warning count maintained by the SOM Compiler.

Syntax

void somtwarn (string *file,* **long** *lineno,* **string** *format, ...);*

Description

The **somtwarn** function prints a warning message and increments the warning count maintained by the SOM Compiler. The message begins with the string “warning: ” and includes the name of the file and the line number on which the error occurred, if specified.

Parameters

file	The name of the file in which the error occurred, or NULL.
lineno	The line number on which the error occurred, or zero.
format	A format string suitable for passing to the **printf** C library function.
varargs	The arguments to be passed to **printf.**

Return Value

None.

Example

```
somtwarn(__get_somtSourceFileName(cls),
         __get_somtSourceLineNumber(entry),
         ”I’m worried about the entry named %s.\n”,
         __get_somtEntryName(entry));
```

Related Information

Functions: somtmsg, somterror, somtfatal, somtinternal

Index

A

B

C

D

E

F

G

L

M

N

O

P

R

S

T

U

W