somf_TSequence Class

Description

The **somf_TSequence** class is an abstract superclass for collections whose objects are ordered.

When you link, include the following library reference to get access to this class: **somtk**

When creating a collection whose objects are ordered, your classes should inherit from **somf_TSequence**. (When creating an unordered collection, your classes should inherit from **somf_TCollection.**) The **somf_TSequence** class's pure virtual functions provide the framework for the methods that should be available in an ordered collection.

File Stem

tseq

Base

somf_TCollection

Metaclass

SOMClass

Ancestor Classes

somf_TCollection, somf_MCollectible, SOMObject

New Methods

somfFirst
somfAfter
somfBefore
somfLast
somfOccurrencesOf
somfTSequenceInit

Overriding Methods

somfAdd
somfRemove
somfRemoveAll
somfDeleteAll
somfCount
somfCreateIterator
somInit

somfAdd Method

Purpose

Adds an object to a given ordered collection.

IDL Syntax

somf_MCollectible somfAdd (in somf_MCollectible *obj*);

Description

The **somfAdd** method adds a specified object `obj` to the ordered collection represented by the receiving object.

Every class that inherits from the **somf_TSequence** class *must* override this method for that class to work correctly.

Parameters

receiver A pointer to an object of class **somf_TSequence.**

ev A pointer to the **Environment** structure for the calling method.

obj A pointer to a **somf_MCollectible** object that will be added to the receiving object.

Return Value

There are two possible valid return values for this method:

somf_MCollectible
A pointer to the **somf_MCollectible** object that had to be removed in order to add `obj`. (Recall that some of the main collection classes will only accept one occurrence of an object where the **somfIsEqual** or **somfIsSame** method would be TRUE.)

SOMF_NIL No **somf_MCollectible** object had to be removed in order to add `obj`.

Example

You cannot use this method directly from this class; it must be overridden. If you invoke this method directly, an error message is written and processing will end. For examples of how this method looks when it is invoked, see **somf_TDeque** or **somf_TSortedSequence**.

Original Class

somf_TCollection (overridden here)

somfAfter Method

Purpose

Gets the object found after a given object `obj` in an ordered collection.

IDL Syntax

somf_MCollectible somfAfter (in somf_MCollectible *obj***);**

Description

The **somfAfter** method returns the object found after object *obj* in the ordered collection represented by the receiving object.

Every class that inherits from the **somf_TSequence** class *must* override this method for that class to work correctly.

Parameters

receiver	A pointer to an object of class **somf_TSequence.**
ev	A pointer to the **Environment** structure for the calling method.
obj	A pointer to the **somf_MCollectible** object that is in front of the returned obj.

Return Value

There are two possible valid return values for this method:

somf_MCollectible
A pointer to the **somf_MCollectible** object after *obj*.

SOMF_NIL The *obj* is the last object in this collection or could not be found.

Example

You cannot use this method directly from this class; it must be overridden. If you invoke this method directly, an error message is written and processing will end. For examples of how this method is invoked, see **somf_TDeque** or **somf_TSortedSequence**.

Original Class

somf_TSequence

Related Information

Methods: somfBefore, somfFirst, somfLast

somfBefore Method

Purpose

Gets the object found before a given `obj` in an ordered collection.

IDL Syntax

somf_MCollectible somfBefore (in somf_MCollectible *obj***);**

Description

The **somfBefore** method returns the object found immediately before the specified object *obj* in the ordered collection represented by the receiving object.

Every class that inherits from the **somf_TSequence** class *must* override this method for that class to work correctly.

Parameters

receiver A pointer to an object of class **somf_TSequence.**

ev A pointer to the **Environment** structure for the calling method.

obj A pointer to the **somf_MCollectible** object that is behind the returned object.

Return Value

There are two possible valid return values for this method:

somf_MCollectible
A pointer to the **somf_MCollectible** object that precedes `obj`.

SOMF_NIL The `obj` is the first object in this collection or could not be found.

Example

You cannot use this method directly from this class; it must be overridden. If you invoke this method directly, an error message is written and processing will end. For examples of how this method is invoked, see **somf_TDeque** or **somf_TSortedSequence**.

Original Class

somf_TSequence

Related Information

Methods: somfAfter, somfFirst, somfLast

somfCount Method

Purpose

Gets the number of objects in this ordered collection.

IDL Syntax

long somfCount ( );

Description

The **somfCount** method determines the number of objects in the ordered collection represented by the receiving object, and returns that number.

Every class that inherits from the **somf_TSequence** class *must* override this method for that class to work correctly.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Thus, if any child of **somf_TCollection** is used with a child of **somf_THashTable**, then the name of the method will have to be fully qualified (for example: **somf_TSequence_somfCount**). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you could have referenced this method as:

```
d->somfCount(ev);
```

Parameters

receiver	A pointer to an object of class **somf_TSequence.**
ev	A pointer to the **Environment** structure for the calling method.

Return Value

This method returns the number of objects in the ordered collection.

Example

You cannot use this method directly from this class; it must be overridden. If you invoke this method directly, an error message is written and processing will end. For examples of how this method is invoked, see **somf_TDeque** or **somf_TSortedSequence**.

Original Class

somf_TCollection (overridden here)

somfCreateIterator Method

Purpose

Returns a new iterator that is suitable for iterating over the objects in an ordered collection.

IDL Syntax

somf_TIterator somfCreateIterator ( );

Description

The **somfCreateIterator** method returns a new iterator that is suitable for iterating over the objects in the ordered collection represented by the receiving object.

Every class that inherits from the **somf_TSequence** class *must* override this method for that class to work correctly.

Parameters

receiver A pointer to an object of class **somf_TSequence.**

ev A pointer to the **Environment** structure for the calling method.

Return Value

This method returns a pointer to the new iterator.

Example

You cannot use this method directly from this class; it must be overridden. If you invoke this method directly, an error message is written and processing will end. For examples of how this method is invoked, see **somf_TDeque** or **somf_TSortedSequence**.

Original Class

somf_TCollection (overridden here)

somfDeleteAll Method

Purpose

Removes all of the objects from an ordered collection and deallocates the storage that these objects might have owned. (That is, the destructor function is called for each object in the collection.)

IDL Syntax

void somfDeleteAll ( );

Description

The **somfDeleteAll** method removes all of the objects from the ordered collection represented by the receiving object. The method also deallocates the storage that these objects might have owned (that is, the destructor function is called for each object in the collection).

Every class that inherits from the **somf_TSequence** class *must* override this method for that class to work correctly.

Be careful with **somfDeleteAll**. Since a collection only contains *pointers* to objects (rather than the objects themselves), **somfDeleteAll** can cause a problem if a pointer to an object appears more than once. For example, if multiple pointers to 'A' exists, or if a single pointer to 'A' is in the collection multiple times, the behavior of the code is undefined, because it will try to delete 'A' multiple times. If you think there is a chance that an object could appear in the collection more than once, you should consider using **somfRemoveAll** to remove the objects from the collection and deleting them some other way.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Thus, if any child of **somf_TCollection** is used with **somf_THashTable**, then the name of the method will have to be fully qualified (for example: **somf_TDictionary_somfDeleteAll**). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you could have referenced this method as:

```
d->somfDeleteAll(ev);
```

Parameters

receiver	A pointer to an object of class **somf_TSequence.**
ev	A pointer to the **Environment** structure for the calling method.

Return Value

None.

Example

You cannot use this method directly from this class; it must be overridden. If you invoke this method directly, an error message is written and processing will end. For examples of how this method is invoked, see **somf_TDeque** or **somf_TSortedSequence**.

Original Class

somf_TCollection (overridden here)

somfFirst Method

Purpose

Gets the first object in an ordered collection.

IDL Syntax

somf_MCollectible somfFirst ( );

Description

The **somfFirst** method determines the first object in the ordered collection represented by the receiving object, and returns a pointer to it.

Every class that inherits from the **somf_TSequence** class *must* override this method for that class to work correctly.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. **somfFirst** is a method name declared in multiple parents (for example: **somf_TSequence**, **somf_TIterator**, etc.). You will probably have to fully qualify the method name (for example: **somf_TDeque_somfFirst**). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you could have referenced this method as:

```
seq->somfFirst(ev);
```

Parameters

receiver	A pointer to an object of class **somf_TSequence.**
ev	A pointer to the **Environment** structure for the calling method.

Return Value

There are two possible valid return values for this method:

somf_MCollectible
: A pointer to the first **somf_MCollectible** object in the ordered collection.

SOMF_NIL
: Nothing is in the collection.

Example

You cannot use this method directly from this class; it must be overridden. If you invoke this method directly, an error message is written and processing will end. For examples of how this method is invoked, see **somf_TDeque** or **somf_TSortedSequence**.

Original Class

somf_TSequence

Related Information

Methods: somfLast, somfAfter, somfBefore

somfLast Method

Purpose

Gets the last object in an ordered collection.

IDL Syntax

somf_MCollectible somfLast ( );

Description

The **somfLast** method determines the last object in the ordered collection represented by the receiving object, and returns a pointer to it.

Every class that inherits from the **somf_TSequence** class *must* override this method for that class to work correctly.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. **somfLast** is a method name declared in multiple parents (for example: **somf_TSequenceIterator**, **somf_TSequence**, etc.). You will probably have to fully qualify the method name (for example: **somf_TDeque_somfLast**). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you could have referenced this method as:

```
seq->somfLast(ev);
```

Parameters

receiver	A pointer to an object of class **somf_TSequence.**
ev	A pointer to the **Environment** structure for the calling method.

Return Value

There are two possible valid return values for this method:

somf_MCollectible
: A pointer to the last **somf_MCollectible** object in the ordered collection.

SOMF_NIL
: Nothing is in the collection.

Example

You cannot use this method directly from this class; it must be overridden. If you invoke this method directly, an error message is written and processing will end. For examples of how this method is invoked, see **somf_TDeque** or **somf_TSortedSequence**.

Original Class

somf_TSequence

Related Information

Methods: somfFirst, somfAfter, somfBefore

somfOccurrencesOf Method

Purpose

Determines the number of times an object `obj` is contained in an ordered collection.

IDL Syntax

long somfOccurrencesOf (in somf_MCollectible *obj***);**

Description

The **somfOccurrencesOf** method determines the number of times a specified object *obj* is contained in an ordered collection represented by the receiving object, and returns that number.

Parameters

receiver	A pointer to an object of class **somf_TSequence.**
ev	A pointer to the **Environment** structure for the calling method.
obj	A pointer to the **somf_MCollectible** object to look for in the collection.

Return Value

This method returns a number indicating how many times *obj* occurs in the collection.

Example

```
somf_TDeque dq;
<your Class which inherits from somf_MCollectible> obj;
Environment *ev;

ev = somGetGlobalEnvironment();

dq = somf_TDequeNew();
obj = <your Class which inherits from somf_MCollectible>New();

_somfAddFirst(dq, ev, obj);

somPrintf("\n There are %d OccurrencesOf obj\n",
                            _somfOccurrencesOf(dq, ev, obj));

_somFree (dq);
_somFree (obj);
```

Original Class

somf_TSequence

somfRemove Method

Purpose

Removes an object from an ordered collection.

IDL Syntax

somf_MCollectible somfRemove (in somf_MCollectible *obj***);**

Description

The **somfRemove** method removes a specified object *obj* from the ordered collection represented by the receiving object.

Every class that inherits from the **somf_TSequence** class *must* override this method for that class to work correctly.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. **somfRemove** is a method name declared in multiple parents (for example: **somf_TCollection**, **somf_THashTable**, **somf_TIterator**, etc.) You will probably have to fully qualify the method name (for example: **somf_TDictionary_somfRemove**). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you could have referenced this method as:

```
d->somfRemove(ev, obj);
```

Parameters

receiver	A pointer to an object of class **somf_TSequence.**
ev	A pointer to the **Environment** structure for the calling method.
obj	A pointer to the **somf_MCollectible** object to be removed from the collection.

Return Value

There are two possible valid return values for this method:

somf_MCollectible
A pointer to the object which was removed.

SOMF_NIL The object was not found.

Example

You cannot use this method directly from this class; it must be overridden. If you invoke this method directly, an error message is written and processing will end. For examples of how this method is invoked, see **somf_TDeque** or **somf_TSortedSequence**.

Original Class

somf_TCollection (overridden here)

Related Information

Methods: somfRemoveAll

somfRemoveAll Method

Purpose

Removes all of the objects from an ordered collection.

IDL Syntax

void somfRemoveAll ( );

Description

The **somfRemoveAll** method removes all of the objects from the ordered collection represented by the receiving object.

Every class that inherits from the **somf_TSequence** class *must* override this method for that class to work correctly.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Thus, if any child of **somf_TCollection** is used with **somf_THashTable**, then the name of the method will have to be fully qualified (for example: **somf_TDictionary_somfRemoveAll**). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you could have referenced this method as:

```
d->somfRemoveAll(ev);
```

Parameters

receiver A pointer to an object of class **somf_TSequence.**

ev A pointer to the **Environment** structure for the calling method.

Return Value

None.

Example

You cannot use this method directly from this class; it must be overridden. If you invoke this method directly, an error message is written and processing will end. For examples of how this method is invoked, see **somf_TDeque** or **somf_TSortedSequence**.

Original Class

somf_TCollection (overridden here)

Related Information

Methods: somfRemove

somfTSequenceInit Method

Purpose

Initializes a new ordered collection of class **somf_TSequence**, given a comparison method for the collection to use.

IDL Syntax

somf_TSequence somfTSequenceInit (in somf_MCollectibleCompareFn *testfn***);**

Description

The **somfTSequenceInit** method initializes the new ordered collection of class **somf_TSequence**, as represented by the receiving object. The method also establishes the comparison method that the new ordered collection will use to compare current/potential objects for the collection, as determined by the *testfn* argument.

Note: You cannot override this method.

Parameters

receiver A pointer to an object of class **somf_TSequence.**

ev A pointer to the **Environment** structure for the calling method.

testfn A method pointer specifying either a **somfIsEqual** or **somfIsSame** method.

This argument should always be set to either
`somf_MCollectibleClassData.somfIsSame` or
`somf_MCollectibleClassData.somfIsEqual`.
This specification is necessary because SOM needs a pointer to the original declaration of the method, which resides in **somf_MCollectible**. The **somf_TSequence** object will use this pointer to access the **somfIsSame** or **somfIsEqual** method that was declared and defined in the object being inserted into, or removed from, the **somf_TSequence** object.

Return Value

This method returns a pointer to an initialized **somf_TSequence** object.

Original Class

somf_TSequence

somf_TSequenceIterator Class

Description

The **somf_TSequenceIterator** class is an abstract base class that defines an iterator for the abstract base class **somf_TSequence**. The methods defined in **somf_TSequenceIterator** will iterate over all of the objects in a sequence.

When you link, include the following library reference to get access to this class: **somtk**

When creating an iterator for an ordered collection, your classes should inherit from the **somf_TSequenceIterator** class. (When creating an iterator for an unordered collection, your classes should inherit from the **somf_TIterator** class.) The **somf_TSequenceIterator** class's pure virtual functions provide the framework for the methods that should be available in an iterator for an ordered collection.

File Stem

tseqitr

Base

somf_TIterator

Metaclass

SOMClass

Ancestor Classes

somf_TIterator, SOMObject

New Methods

somfLast
somfPrevious

Overriding Methods

somfFirst
somfNext
somfRemove

somfFirst Method

Purpose

Resets the iterator and gets the first object of an ordered collection.

IDL Syntax

somf_MCollectible somfFirst ( );

Description

The **somfFirst** method resets the iterator and returns the first object of the ordered collection that corresponds to the iterator used as the receiving object.

The **somfFirst** method resets the iterator to the beginning of the collection. This is true not only for the first time the iterator is used; it is also true if other operations on the collection cause the iterator to be invalidated. In the second case, the method also revalidates the iterator.

Every class that inherits from the **somf_TSequenceIterator** class *must* override this method for that class to work correctly.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. **somfFirst** is a method name declared in multiple parents (for example: **somf_TSequence**, **somf_TIterator**, etc.). You will probably have to fully qualify the method name (for example: **somf_TDequeIterator_somfFirst**). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you could have referenced this method as:

```
itr->somfFirst(ev);
```

Parameters

receiver	A pointer to an object of class **somf_TSequenceIterator.**
ev	A pointer to the **Environment** structure for the calling method.

Return Value

This method returns a pointer to the first **somf_MCollectible** object in the ordered collection.

Example

You cannot use this method directly from this class; it must be overridden. If you invoke this method directly, an error message is written and processing will end. For examples of how this method is invoked, see **somf_TDequeIterator** or **somf_TSortedSequenceIterator**.

Original Class

somf_TIterator (overridden here)

Related Information

Methods: somfNext

somfLast Method

Purpose

Gets the last object in an ordered collection.

IDL Syntax

somf_MCollectible somfLast ( );

Description

The **somfLast** method determines the last object in the **somf_TSequence** collection that corresponds to the **somf_TSequenceIterator** iterator used as the receiving object, and returns a pointer to it.

Every class that inherits from the **somf_TSequenceIterator** class *must* override this method for that class to work correctly.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. **somfLast** is a method name declared in multiple parents (for example: **somf_TSequenceIterator**, **somf_TSequence**, etc.). You will probably have to fully qualify the method name (for example: **somf_TSortedSequenceIterator_somfLast**). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you could have referenced this method as:

```
itr->somfLast(ev);
```

Parameters

receiver	A pointer to an object of class **somf_TSequenceIterator.**
ev	A pointer to the **Environment** structure for the calling method.

Return Value

This method returns a pointer to the last **somf_MCollectible** in the collection.

Example

You cannot use this method directly from this class; it must be overridden. If you invoke this method directly, an error message is written and processing will end. For examples of how this method is invoked, see **somf_TDequeIterator** or **somf_TSortedSequenceIterator**.

Original Class

somf_TSequenceIterator

Related Information

Methods: somfPrevious

somfNext Method

Purpose

Gets the next object in an ordered collection.

IDL Syntax

somf_MCollectible somfNext ( );

Description

The **somfNext** method determines the next object in the ordered collection that corresponds to the iterator used as the receiving object. The method also returns a pointer to the next object, if found. Objects are retrieved in an order that reflects the "ordered–ness" of the collection (or the lack of ordering on the collection elements).

Every class that inherits from the **somf_TSequenceIterator** class *must* override this method for that class to work correctly.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Thus, if any child of **somf_TIterator** is used with a child of **somf_TPrimitiveLinkedListIterator**, then the name of the method will have to be fully qualified (for example: **somf_TDictionaryIterator_somfNext**). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you could have referenced this method as:

```
itr->somfNext(ev);
```

If the collection has changed since the last time **somfFirst** was called (other than through the use of the **somfRemove** method of this iterator), this method will *fail.*

Parameters

receiver	A pointer to an object of class **somf_TSequenceIterator.**
ev	A pointer to the **Environment** structure for the calling method.

Return Value

There are two possible valid return values for this method:

somf_MCollectible
A pointer to the next **somf_MCollectible** object in the collection.

SOMF_NIL The end of the collection has been reached.

Example

You cannot use this method directly from this class; it must be overridden. If you invoke this method directly, an error message is written and processing will end. For examples of how this method is invoked, see **somf_TDequeIterator** or **somf_TSortedSequenceIterator**.

Original Class

somf_TIterator (overridden here)

Related Information

Methods: somfFirst

somfPrevious Method

Purpose

Gets the previous object in an ordered collection.

IDL Syntax

somf_MCollectible somfPrevious ( );

Description

The **somfPrevious** method determines the previous object in the **somf_TSequence** collection that corresponds to the **somf_TSequenceIterator** iterator used as the receiving object, and returns a pointer to the previous object (if found).

Every class that inherits from the **somf_TSequenceIterator** class *must* override this method for that class to work correctly.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Thus, if any child of **somf_TSequenceIterator** is used with **somf_TPrimitiveLinkedListIterator**, then the name of the method will have to be fully qualified (for example: **somf_TSortedSequenceIterator_somfPrevious**). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you could have referenced this method as:

```
itr->somfPrevious(ev);
```

Parameters

receiver A pointer to an object of class **somf_TSequenceIterator.**

ev A pointer to the **Environment** structure for the calling method.

Return Value

There are two possible valid return values for this method:

somf_MCollectible
A pointer to the previous **somf_MCollectible** object in the collection.

SOMF_NIL The beginning of the collection has been reached.

Example

You cannot use this method directly from this class; it must be overridden. If you invoke this method directly, an error message is written and processing will end. For examples of how this method is invoked, see **somf_TDequeIterator** or **somf_TSortedSequenceIterator**.

Original Class

somf_TSequenceIterator

Related Information

Methods: somfLast

somfRemove Method

Purpose

Removes the current object from an ordered collection.

IDL Syntax

void somfRemove ( );

Description

The **somfRemove** method removes the current object (the one just returned by **somfFirst, somfNext, somfLast,** or **somfPrevious**) from the ordered collection that corresponds to the iterator used as the receiving object.

The **somfRemove** method is the only way to remove an object from an ordered collection during iteration. However, if multiple iterators are in process, all other iterators are invalidated, just as if some other kind of change had occurred in the collection

If the collection has changed (other than through the use of the **somfRemove** method of this iterator) since the last time **somfFirst** or **somfLast** was called, this method will *fail*.

Every class that inherits from the **somf_TSequenceIterator** class *must* override this method for that class to work correctly.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. **somfRemove** is a method name declared in multiple parents (for example: **somf_TCollection**, **somf_THashTable**, **somf_TIterator**, etc.) You will probably have to fully qualify the method name (as **somf_TDictionaryIterator_somfRemove,** for example). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you could have referenced this method as:

```
itr->somfRemove(ev);
```

Parameters

receiver	A pointer to an object of class **somf_TSequenceIterator.**
ev	A pointer to the **Environment** structure for the calling method.

Return Value

None.

Example

You cannot use this method directly from this class; it must be overridden. If you invoke this method directly, an error message is written and processing will end. For examples of how this method is invoked, see **somf_TDequeIterator** or **somf_TSortedSequenceIterator**.

Original Class

somf_TIterator (overridden here)

somf_TSet Class

Description

The **somf_TSet** class is a subclass of **somf_TCollection**. It represents an unordered collection of objects in which objects can appear only once.

When you link, include the following library reference to get access to this class: **somtk**

Because **somf_TSet** takes objects of the **somf_MCollectible** class as members, any class that inherits from **somf_MCollectible** can be an element of the set. This means, for example, that you can have a set containing **somf_TDeque** objects, or a set of **somf_TDictionary** objects, or objects of any main collection class.

Objects that are inserted into the **somf_TSet** collection *must* inherit from **somf_MCollectible**. In addition, they *must* override the **somfHash** method, and the **somfIsEqual** method. These are used internally by collections of the **somf_TSet** class.

Note: The **somf_TSet** class uses the **somfIsEqual** method as the default comparison function. (That is, if `key1="Bart"` and `key2="Bart"`, then `key1` and `key2` are equal.) If you do not want to use the **somfIsEqual** method to equate entries, use one of the initialization methods to change to the **somfIsSame** method.

Warning: The **somf_TSet** class only allows objects to be in the collection once. If an object will be needed in the set more than once, you should consider using a **somf_TDeque** instead.

Although the methods in this class are reentrant, the class is not thread-safe on multi-thread applications. If a pointer to an instance of this class will be passed to multiple threads, the code in those threads must guarantee thread-safe usage of the class.

File Stem

tset

Base

somf_TCollection

Metaclass

SOMClass

Ancestor Classes

somf_TCollection, somf_MCollectible, SOMObject

New Methods

somfDifferenceS
somfDifferenceSS
somfIntersectionS
somfIntersectionSS
somfUnionS
somfUnionSS
somfXorS
somfXorSS
somfSetHashFunction
somfGetHashFunction
somfRehash

somfAssign
somfTSetInitFL
somfTSetInitF
somfTSetInitLF
somfTSetInitL
somfTSetInitS

Overriding Methods

somInit
somUninit
somfAdd
somfRemove
somfRemoveAll
somfDeleteAll
somfCount
somfMember
somfCreateIterator

somfAdd Method

Purpose

Adds an object to a given set.

IDL Syntax

somf_MCollectible somfAdd (in somf_MCollectible *obj***);**

Description

The **somfAdd** method adds the specified object *obj* to the set used as the receiving object.

Parameters

receiver	A pointer to an object of class **somf_TSet.**
ev	A pointer to the **Environment** structure for the calling method.
obj	A pointer to the **somf_MCollectible** object that will be added to the set.

Return Value

There are two possible valid return values for this method:

somf_MCollectible
: A pointer to the **somf_MCollectible** object that had to be removed in order to add *obj*.

SOMF_NIL
: No **somf_MCollectible** object had to be removed in order to add *obj*.

Example

```
somf_TSet s;
<Your class which inherits from somf_MCollectible> obj;
Environment *ev;

ev = somGetGlobalEnvironment();

s = somf_TSetNew();
obj = <Your class which inherits from somf_MCollectible>New();

/* Add obj to s */
if (_somfAdd(s, ev, obj) != SOMF_NIL)
    somPrintf("\n problem adding obj to s\n");

_somFree (s);
```

Original Class

somf_TCollection (overridden here)

somfAssign Method

Purpose

Assigns a set as being equal to a given source set.

IDL Syntax

void somfAssign (in somf_TSet *source***);**

Description

The **somfAssign** method assigns the set used as the receiving object to be equal to the source set. That is, the method sets/resets the instance variables of the receiver to the values of the source. This operation is logically equivalent to using the "=" operator.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Thus, if any child of **somf_TSet** is used with any other main collection class, then the name of the method will have to be fully qualified (for example: **somf_TSet_somfAssign**). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you could have referenced this method as:

```
d->somfAssign(ev, d2);
```

Parameters

receiver	A pointer to an object of class **somf_TSet.**
ev	A pointer to the **Environment** structure for the calling method.
source	A pointer to the set to which the receiving object will be made equal.

Return Value

None.

Example

```
somf_TSet s1;
somf_TSet s2;
Environment *ev;

ev = somGetGlobalEnvironment();

s1 = somf_TSetNew();
s2 = somf_TSetNew();

/* Add som objects to s1 */

/* Assign s2 = s1 */
somf_TSet_somfAssign(s2, ev, s1);

_somFree (s1);
_somFree (s2);
```

Original Class

somf_TSet

somfCount Method

Purpose

Gets the number of objects in a given set.

IDL Syntax

long somfCount ( );

Description

The **somfCount** method determines the number of objects in the set used as the receiving object, and returns that number.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Thus, if any child of **somf_TCollection** is used with **somf_THashTable**, then the name of the method will have to be fully qualified (for example: **somf_TDictionary_somfCount**). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you could have referenced this method as:

```
d->somfCount(ev);
```

Parameters

receiver A pointer to an object of class **somf_TSet.**

ev A pointer to the **Environment** structure for the calling method.

Return Value

This method returns the number of objects contained in the set.

Example

```
somf_TSet s;
Environment *ev;

ev = somGetGlobalEnvironment();

s = somf_TSetNew();

somPrintf("\n Count of s= %d\n", _somfCount(s,ev));

_somFree (s);
```

Original Class

somf_TCollection (overridden here)

somfCreateIterator Method

Purpose

Returns a new iterator that is suitable for iterating over the objects in this set.

IDL Syntax

somf_TIterator somfCreateIterator ( );

Description

The **somfCreateIterator** method returns a new iterator that is suitable for iterating over the objects in this set.

Note: This is one of two ways to initialize a **somf_TSetIterator** iterator to point to an instance of **somf_TSet**. The other way is to use the **somf_TSetIterator** class's initializer method described on page 299.

Parameters

receiver A pointer to an object of class **somf_TSet.**

ev A pointer to the **Environment** structure for the calling method.

Return Value

This method returns a pointer to the new iterator.

Example

```
somf_TSet s;
Environment *ev;
somf_TSetIterator itr;

ev = somGetGlobalEnvironment();

s = somf_TSetNew();
itr = (somf_TSetIterator*) _somfCreateIterator(s,ev);

_somFree (s);
_somFree (itr);
```

Original Class

somf_TCollection (overridden here)

somfDeleteAll Method

Purpose

Removes all of the objects from a set and deallocates the storage that these objects might have owned. (That is, the destructor function is called for each object in the set.)

IDL Syntax

void somfDeleteAll ( );

Description

The **somfDeleteAll** method removes all of the objects from the set represented by the receiving object. The method also deallocates the storage that these objects might have owned (that is, the destructor function is called for each object in the collection).

Be careful with **somfDeleteAll**. Since a collection only contains *pointers* to objects (rather than the objects themselves), **somfDeleteAll** can cause a problem if a pointer to an object appears more than once. For example, if multiple pointers to 'A' exist, or if a single pointer to 'A' is in the collection multiple times, the behavior of the code is undefined, because it will try to delete 'A' multiple times. If you think there is a chance that an object could appear in the collection more than once, you should consider using **somfRemoveAll** to remove the objects from the collection and deleting them some other way.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Thus, if any child of **somf_TCollection** is used with **somf_THashTable**, then the name of the method will have to be fully qualified (for example: **somf_TSet_somfDeleteAll**). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you could have referenced this method as:

```
d->somfDeleteAll(ev);
```

Parameters

receiver	A pointer to an object of class **somf_TSet.**
ev	A pointer to the **Environment** structure for the calling method.

Return Value

None.

Example

```
somf_TSet s;
Environment *ev;

ev = somGetGlobalEnvironment();

s = somf_TSetNew();

/* Add some objects to s */

/* Remove all of the objects from s AND DELETE THEM */
_somfDeleteAll(s,ev);

_somFree (s);
```

Original Class

somf_TCollection (overridden here)

somfDifferenceS Method

Purpose

Determines the elements of a given set that do not appear in another specified set, and modifies the first set to contain only those different elements.

IDL Syntax

void somfDifferenceS (in somf_TSet *set1***);**

Description

The **somfDifferenceS** method determines those elements of a given set that are not contained in another specified set, *set1*. The set used as the receiving object is destructively modified to contain only those elements that do not appear in *set1*.

Parameters

receiver	A pointer to an object of class **somf_TSet.**
ev	A pointer to the **Environment** structure for the calling method.
set1	A pointer to the set that the receiving object will be compared against.

Return Value

None.

Example

```
somf_TSet s1;
somf_TSet s2;
Environment *ev;

ev = somGetGlobalEnvironment();

s1 = somf_TSetNew();
s2 = somf_TSetNew();

/* Find the differences between s1 and s2, and put it in s1 */
_somfDifferenceS(s1, ev, s2);

_somFree (s1);
_somFree (s2);
```

Original Class

somf_TSet

Related Information

Methods: somfDifferenceSS

somfDifferenceSS Method

Purpose

Determines the elements of a given set that do not appear in another specified set, and places those different elements in a third set.

IDL Syntax

void somfDifferenceSS (
in somf_TSet *set1***,**
in somf_TSet *resultSet***);**

Description

The **somfDifferenceSS** method determines which elements of the receiving-object set are not contained in set *set1*. Those elements that do not appear in set *set1* are then placed in set *resultSet.* The original receiving-object set remains unchanged.

Parameters

receiver	A pointer to an object of class **somf_TSet.**
ev	A pointer to the **Environment** structure for the calling method.
set1	A pointer to the set that the receiving object set will be compared against.
resultSet	A pointer to the set containing the results of the operation.

Return Value

None.

Example

```
somf_TSet s1;
somf_TSet s2;
somf_TSet s3;
Environment *ev;

ev = somGetGlobalEnvironment();

s1 = somf_TSetNew();
s2 = somf_TSetNew();
s3 = somf_TSetNew();

/* Find the differences between s1 and s2, and put it in s3 */
_somfDifferenceSS(s1, ev, s2, s3);

_somFree (s1);
_somFree (s2);
_somFree (s3);
```

Original Class

somf_TSet

Related Information

Methods: somfDifferenceS

somfGetHashFunction Method

Purpose

Gets a pointer to the hash function used by a set.

IDL Syntax

somf_MCollectibleHashFn somfGetHashFunction ( );

Description

The **somfGetHashFunction** method returns a pointer to the hash function used by the set represented by the receiving object.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Thus, if **somf_TSet** is used with a child of **somf_THashTable** or **somf_TDictionary**, then the name of the method will have to be fully qualified (for example: **somf_TSet_somfGetHashFunction**). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you could have referenced this method as:

```
d->somfGetHashFunction(ev);
```

Parameters

receiver	A pointer to an object of class **somf_TSet.**
ev	A pointer to the **Environment** structure for the calling method.

Return Value

This method returns a pointer to the hash function.

Example

```
somf_TSet s;
Environment *ev;

ev = somGetGlobalEnvironment();

s = somf_TSetNew();

if ((somf_TSet_somfGetHashFunction(s,ev)) !=
              somf_MCollectibleClassData.somfHash)
    somPrintf("\n What Hash Function are we using?\n");

_somFree (s);
```

Original Class

somf_TSet

Related Information

Methods: somfSetHashFunction

somfIntersectionS Method

Purpose

Gets those elements that are members of both a given set and another set, `set1`, and modifies the first set to contain only those common elements.

IDL Syntax

void somfIntersectionS (in somf_TSet *set1***);**

Description

The **somfIntersectionS** method determines, given the receiving-object set and *set1,* which elements are contained in both sets. The set used as the receiving object is then destructively modified to contain only those common elements.

Parameters

receiver	A pointer to an object of class **somf_TSet.**
ev	A pointer to the **Environment** structure for the calling method.
set1	A pointer to the set that the receiving object will be compared against.

Return Value

None.

Example

```
somf_TSet s1;
somf_TSet s2;
Environment *ev;

ev = somGetGlobalEnvironment();

s1 = somf_TSetNew();
s2 = somf_TSetNew();

/* Find the intersection between s1 and s2, and put it in s1 */
_somfIntersectionS(s1, ev, s2);

_somFree (s1);
_somFree (s2);
```

Original Class

somf_TSet

Related Information

Methods: somfIntersectionSS

somfIntersectionSS Method

Purpose

Gets those elements that are members of both a given set and another set, `set1`, and places those common elements in a third set.

IDL Syntax

void somfIntersectionSS (
in somf_TSet *set1***,**
in somf_TSet *resultSet***);**

Description

The **somfIntersectionSS** method determines, given the receiving-object set and *set1,* which elements are contained in both sets. The common elements are then placed in set *resultSet.* The original receiving-object set and *set1* remain unchanged.

Parameters

receiver	A pointer to an object of class **somf_TSet.**
ev	A pointer to the **Environment** structure for the calling method.
set1	A pointer to the set this instance will be compared against.
resultSet	A pointer to the set containing the results of the operation.

Return Value

None.

Example

```
somf_TSet s1;
somf_TSet s2;
somf_TSet s3;
Environment *ev;

ev = somGetGlobalEnvironment();

s1 = somf_TSetNew();
s2 = somf_TSetNew();
s3 = somf_TSetNew();

/* Find the intersection between s1 and s2, and put it in s3 */
_somfIntersectionSS(s1, ev, s2, s3);

_somFree (s1);
_somFree (s2);
_somFree (s3);
```

Original Class

somf_TSet

Related Information

Methods: somfIntersectionS

somfMember Method

Purpose

Gets an object from a set.

IDL Syntax

somf_MCollectible somfMember (in somf_MCollectible *obj*);

Description

The **somfMember** method determines whether the specified object *obj* is a member of the set represented by the receiving object, and returns a pointer to the object (if found).

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Thus, if any child of **somf_TCollection** is used with **somf_THashTable**, then the name of the method will have to be fully qualified (for example: **somf_TSet_somfMember**). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you could have referenced this method as:

```
d->somfMember(ev, obj);
```

Parameters

receiver	A pointer to an object of class **somf_TSet.**
ev	A pointer to the **Environment** structure for the calling method.
obj	A pointer to the **somf_MCollectible** that may or may not be a member of the Collection.

Return Value

There are two possible valid return values for this method:

somf_MCollectible
: A pointer to the object the method determined as the member.

SOMF_NIL
: The object was not found.

Example

```
somf_TSet s;
<your Class which inherits from somf_MCollectible> obj;
Environment *ev;

ev = somGetGlobalEnvironment();

s = somf_TSetNew();
obj = <your Class which inherits from somf_MCollectible>New();

_somfAdd(s, ev, obj);

if (_somfMember(s, ev, obj) != SOMF_NIL)
    somPrintf("\n obj is a Member\n");
else
    somPrintf("\n ERROR: obj should be a Member\n");

_somFree (s);
_somFree (obj);
```

Original Class

somf_TCollection (overridden here)

somfRehash Method

Purpose

Rehashes a set, cleaning up for any objects that were marked for deletion.

IDL Syntax

void somfRehash ( );

Description

The **somfRehash** method rehashes the set represented by the receiving object, and cleans up for any objects that were marked for deletion.

Note: You cannot override this method.

Parameters

receiver A pointer to an object of class **somf_TSet.**

ev A pointer to the **Environment** structure for the calling method.

Return Value

None.

Example

```
somf_TSet s;
Environment *ev;

ev = somGetGlobalEnvironment();

s = somf_TSetNew();

_somfRehash(s,ev);

_somFree (s);
```

Original Class

somf_TSet

somfRemove Method

Purpose

Removes an object from a given set.

IDL Syntax

somf_MCollectible somfRemove (in somf_MCollectible *obj***);**

Description

The **somfRemove** method removes a specified object *obj* from the set represented by the receiving object.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. **somfRemove** is a method name declared in multiple parents (for example: **somf_TCollection**, **somf_THashTable**, **somf_TIterator**, etc.) You will probably have to fully qualify the method name (for example: **somf_TSet_somfRemove**). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you could have referenced this method as:

```
d->somfRemove(ev, obj);
```

Parameters

receiver	A pointer to an object of class **somf_TSet.**
ev	A pointer to the **Environment** structure for the calling method.
obj	A pointer to the **somf_MCollectible** object to be removed from the set.

Return Value

There are two possible valid return values for this method:

somf_MCollectible
: A pointer to the object that was removed.

SOMF_NIL
: The object was not found.

Example

```
somf_TSet s;
<your Class which inherits from somf_MCollectible> obj;
Environment *ev;

ev = somGetGlobalEnvironment();

s = somf_TSetNew();
obj = <your Class which inherits from somf_MCollectible>New();

_somfAdd(s, ev, obj);
if (somf_TSet_somfRemove(s, ev, obj) == SOMF_NIL)
    somPrintf("\n problem removing obj from s\n");

_somFree (s);
_somFree (obj);
```

Original Class

somf_TCollection (overridden here)

Related Information

Methods: somfRemoveAll

somfRemoveAll Method

Purpose

Removes all of the objects from a given set.

IDL Syntax

void somfRemoveAll ( );

Description

The **somfRemoveAll** method removes all of the objects from the set represented by the receiving object.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Thus, if any child of **somf_TCollection** is used with a child of **somf_THashTable**, then the name of the method will have to be fully qualified (for example: **somf_TSet_somfRemoveAll**). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you could have referenced this method as:

```
d->somfRemoveAll(ev);
```

Parameters

receiver	A pointer to an object of class **somf_TSet.**
ev	A pointer to the **Environment** structure for the calling method.

Return Value

None.

Example

```
somf_TSet s;
Environment *ev;

ev = somGetGlobalEnvironment();

s = somf_TSetNew();

/* Remove All of the objects in s */
_somfRemoveAll(s,ev);

_somFree (s);
```

Original Class

somf_TCollection (overridden here)

Related Information

Methods: somfRemove

somfSetHashFunction Method

Purpose

Sets the hash function of a set.

IDL Syntax

void somfSetHashFunction (in somf_MCollectibleHashFn *fn*);

Description

The **somfSetHashFunction** method sets the hash function of the set used as the receiving object to the specified method *fn*.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Thus, if a child of **somf_TDictionary** is used with a child of **somf_THashTable** or **somf_TSet**, then the name of the method will have to be fully qualified (for example: **somf_TSet_somfSetHashFunction**). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you could have referenced this method as:

```
d->somfSetHashFunction(ev);
```

Parameters

receiver — A pointer to an object of class **somf_TSet.**

ev — A pointer to the **Environment** structure for the calling method.

fn — A method pointer specifying a **somfHash** type method.

This argument should always be set to

```
somf_MCollectibleClassData.somfHash
```

This specification is necessary because SOM needs a pointer to the original declaration of the method, which resides in **somf_MCollectible**. The **somf_TSet** object will use this pointer to access the **somfHash** method that was declared and defined in the object being inserted into, or removed from, the **somf_TSet** object.

Return Value

None.

Example

```
somf_TSet s;
Environment *ev;

ev = somGetGlobalEnvironment();

s = somf_TSetNew();

somf_TSet_somfSetHashFunction(s, ev,
                              somf_MCollectibleClassData.somfHash);

_somFree (s);
```

Original Class

somf_TSet

Related Information

Methods: somfGetHashFunction

somfTSetInitF Method

Purpose

Initializes a new set, given its comparison test method.

IDL Syntax

somf_TSet somfTSetInitF (in somf_MCollectibleCompareFn *testfn***);**

Description

The **somfTSetInitF** method initializes the set represented by the receiving object, given the comparison test method that the set will use. The method assumes a default number of objects as the set size.

Note: You cannot override this method.

Parameters

receiver A pointer to an object of class **somf_TSet.**

ev A pointer to the **Environment** structure for the calling method.

testfn A method pointer specifying either a **somfIsEqual** or **somfIsSame** method.

This argument should always be set to either
`somf_MCollectibleClassData.somfIsSame` or
`somf_MCollectibleClassData.somfIsEqual`.
This specification is necessary because SOM needs a pointer to the original declaration of the method, which resides in **somf_MCollectible**. The **somf_TSet** object will use this pointer to access the **somfIsSame** or **somfIsEqual** method that was declared and defined in the object being inserted into, or removed from, the **somf_TSet** object.

Return Value

This method returns a pointer to an initialized **somf_TSet** object.

Example

```
somf_TSet s2;
Environment *ev;

ev = somGetGlobalEnvironment();

s2 = somf_TSetNew();
_somfTSetInitF(s2, ev, somf_MCollectibleClassData.somfIsEqual);

_somFree (s2);
```

Original Class

somf_TSet

Related Information

Methods: somfTSetInitFL, somfTSetInitLF, somfTSetInitL, somfTSetInitS

somfTSetInitFL Method

Purpose

Initializes a new set, given the comparison test method and the initial set size. Note: This method is equivalent to the method **somfTSetInitLF.**

IDL Syntax

somf_TSet somfTSetInitFL (
in somf_MCollectibleCompareFn *testfn***,**
in long *setSizeHint***);**

Description

The **somfTSetInitFL** method initializes the set represented by the receiving object, given the set's comparison test method and initial set size.

Note: You cannot override this method.

Parameters

receiver A pointer to an object of class **somf_TSet.**

ev A pointer to the **Environment** structure for the calling method.

testfn A method pointer specifying either a **somfIsEqual** or **somfIsSame** method.

This argument should always be set to either
`somf_MCollectibleClassData.somfIsSame` or
`somf_MCollectibleClassData.somfIsEqual.`
This specification is necessary because SOM needs a pointer to the original declaration of the method, which resides in **somf_MCollectible**. The **somf_TSet** object will use this pointer to access the **somfIsSame** or **somfIsEqual** method that was declared and defined in the object being inserted into, or removed from, the **somf_TSet** object.

setSizeHint The initial size of the set, expressed as the number of objects the set is expected to contain.

Return Value

This method returns a pointer to an initialized **somf_TSet** object.

Example

```
somf_TSet s1;
Environment *ev;

ev = somGetGlobalEnvironment();

s1 = somf_TSetNew();
_somfTSetInitFL(s1, ev, somf_MCollectibleClassData.somfIsEqual, 8);

_somFree (s1);
```

Original Class

somf_TSet

Related Information

Methods: somfTSetInitF, somfTSetInitLF, somfTSetInitL, somfTSetInitS

somfTSetInitL Method

Purpose

Initializes a new set, given the initial set size.

IDL Syntax

somf_TSet somfTSetInitL (in long *setSizeHint***);**

Description

The **somfTSetInitL** method initializes the set represented by the receiving object, given the set's initial set size. The method assumes the **somf_TSet** class's default comparison test function of **somfIsEqual.**

Note: You cannot override this method.

Parameters

receiver	A pointer to an object of class **somf_TSet.**
ev	A pointer to the **Environment** structure for the calling method.
setSizeHint	The initial size of the set, expressed as the number of objects the set is expected to contain.

Return Value

This method returns a pointer to an initialized **somf_TSet**.

Example

```
somf_TSet s4;
Environment *ev;

ev = somGetGlobalEnvironment();

s4 = somf_TSetNew();
_somfTSetInitL(s4, ev, 8);

_somFree (s4);
```

Original Class

somf_TSet

Related Information

Methods: somfTSetInitFL, somfTSetInitF, somfTSetInitLF, somfTSetInitS

somfTSetInitLF Method

Purpose

Initializes a new set, given the initial set size and the comparison test method. Note: This method is equivalent to method **somfTSetInitFL.**

IDL Syntax

somf_TSet somfTSetInitLF (
in long *setSizeHint***,**
in somf_MCollectibleCompareFn *testfn***);**

Description

The **somfTSetInitLF** method initializes the set represented by the receiving object, given the set's initial set size and its comparison test method.

Note: You cannot override this method.

Parameters

receiver A pointer to an object of class **somf_TSet.**

ev A pointer to the **Environment** structure for the calling method.

setSizeHint The initial size of the set, expressed as the number of objects the set is expected to contain.

testfn A method pointer specifying either a **somfIsEqual** or **somfIsSame** method.

This argument should always be set to either
`somf_MCollectibleClassData.somfIsSame` or
`somf_MCollectibleClassData.somfIsEqual.`
This specification is necessary because SOM needs a pointer to the original declaration of the method, which resides in **somf_MCollectible**. The **somf_TSet** object will use this pointer to access the **somfIsSame** or **somfIsEqual** method that was declared and defined in the object being inserted into, or removed from, the **somf_TSet** object.

Return Value

This method returns a pointer to an initialized **somf_TSet** object.

Example

```
somf_TSet s3;
Environment *ev;

ev = somGetGlobalEnvironment();

s3 = somf_TSetNew();
_somfTSetInitLF(s3, ev, 8, somf_MCollectibleClassData.somfIsEqual);

_somFree (s3);
```

Original Class

somf_TSet

Related Information

Methods: somfTSetInitFL, somfTSetInitF, somfTSetInitL, somfTSetInitS

somfTSetInitS Method

Purpose

Initializes a new set, establishing it as equal to another given set.

IDL Syntax

somf_TSet somfTSetInitS (in somf_TSet *s*);

Description

The **somfTSetInitS** method initializes the set represented by the receiving object. The method also establishes the new set as equal to the specified source set. This implies that the instance data of the new set will be equal to those of the source set.

Note: You cannot override this method.

Parameters

receiver	A pointer to an object of class **somf_TSet.**
ev	A pointer to the **Environment** structure for the calling method.
s	A pointer to the set to which the receiving object will be equal.

Return Value

This method returns a pointer to an initialized **somf_TSet** object.

Example

```
somf_TSet s4;
somf_TSet s5;
Environment *ev;

ev = somGetGlobalEnvironment();

s4 = somf_TSetNew();
s5 = somf_TSetNew();
_somfTSetInitS(s5, ev, s4);

_somFree (s4);
_somFree (s5);
```

Original Class

somf_TSet

Related Information

Methods: somfTSetInitFL, somfTSetInitF, somfTSetInitLF, somfTSetInitL

somfUnionS Method

Purpose

Gets those elements that are members of either a given set or another set, `set1`, and modifies the first set to contain all elements from both sets.

IDL Syntax

void somfUnionS (in somf_TSet *set1***);**

Description

The **somfUnionS** method determines the set of elements that are contained in either the receiving object set or in the set *set1*. The set used as the receiving object is then destructively modified to contain all of those elements from both sets.

Parameters

receiver	A pointer to an object of class **somf_TSet.**
ev	A pointer to the **Environment** structure for the calling method.
set1	A pointer to the set that the receiving object will be compared against.

Return Value

None.

Example

```
somf_TSet s1;
somf_TSet s2;
Environment *ev;

ev = somGetGlobalEnvironment();

s1 = somf_TSetNew();
s2 = somf_TSetNew();

/* Find the union between s1 and s2, and put it in s1 */
_somfUnionS(s1, ev, s2);

_somFree (s1);
_somFree (s2);
```

Original Class

somf_TSet

Related Information

Methods: somfUnionSS

somfUnionSS Method

Purpose

Gets those elements that are members of either a given set and another set, `set1`, and places all those elements in a third set.

IDL Syntax

void somfUnionSS (
in somf_TSet *set1***,**
in somf_TSet *resultSet***);**

Description

The **somfUnionSS** method determines the set of elements that are contained either in the receiving-object set or in *set1.* All of those elements are then placed in set *resultSet.* The original receiving-object set and *set1* remain unchanged.

Parameters

receiver	A pointer to an object of class **somf_TSet.**
ev	A pointer to the **Environment** structure for the calling method.
set1	A pointer to the set that the receiving object will be compared against.
resultSet	A pointer to the set containing the results of the operation.

Return Value

None.

Example

```
somf_TSet s1;
somf_TSet s2;
somf_TSet s3;
Environment *ev;

ev = somGetGlobalEnvironment();

s1 = somf_TSetNew();
s2 = somf_TSetNew();
s3 = somf_TSetNew();

/* Find the union between s1 and s2, and put it in s3 */
_somfUnionSS(s1, ev, s2, s3);

_somFree (s1);
_somFree (s2);
_somFree (s3);
```

Original Class

somf_TSet

Related Information

Methods: somfUnionS

somfXorS Method

Purpose

Determines a set wherein each member is an element either of a given set or of another set *set1,* but not of both, and modifies the first set to contain the elements of the new set.

IDL Syntax

void somfXorS (in somf_TSet *set1***);**

Description

The **somfXorS** method determines a set wherein each member is an element either of the set represented by the receiving object or of another set, *set1,* but not both. The receiving object set is then modified to contain all of the elements of the newly determined set.

Parameters

receiver	A pointer to an object of class **somf_TSet.**
ev	A pointer to the **Environment** structure for the calling method.
set1	A pointer to the set that the receiving object will be compared against.

Return Value

None.

Example

```
somf_TSet s1;
somf_TSet s2;
Environment *ev;

ev = somGetGlobalEnvironment();

s1 = somf_TSetNew();
s2 = somf_TSetNew();

/* Find the exclusive or of s1 and s2, and put it in s1 */
_somfXorS(s1, ev, s2);

_somFree (s1);
_somFree (s2);
```

Original Class

somf_TSet

Related Information

Methods: somfXorSS

somfXorSS Method

Purpose

Determines a set where each member is an element either of a given set or of another set *set1,* but not of both, and places all of those elements in a third set.

IDL Syntax

void somfXorSS (
in somf_TSet *set1***,**
in somf_TSet *resultSet***);**

Description

The **somfXorSS** method determines a set where each member is an element either of the set represented by the receiving object or of another set, *set1,* but not both. All elements of the newly determined set are then placed in set *resultSet.* The receiving-object set and *set1* remain unchanged.

Parameters

receiver	A pointer to an object of class **somf_TSet.**
ev	A pointer to the **Environment** structure for the calling method.
set1	A pointer to the set that the receiving object will be compared against.
resultSet	A pointer to the set containing the results of the operation.

Return Value

None.

Example

```
somf_TSet s1;
somf_TSet s2;
somf_TSet s3;
Environment *ev;

ev = somGetGlobalEnvironment();

s1 = somf_TSetNew();
s2 = somf_TSetNew();
s3 = somf_TSetNew();

/* Find the exclusive or of s1 and s2, and put it in s3 */
_somfXorSS(s1, ev, s2, s3);

_somFree (s1);
_somFree (s2);
_somFree (s3);
```

Original Class

somf_TSet

Related Information

Methods: somfXorS

somf_TSetIterator Class

Description

The **somf_TSetIterator** class defines an iterator for the **somf_TSet** class that will iterate over all of the objects in a set.

When you link, include the following library reference to get access to this class: **somtk**

Although the methods in this class are reentrant, the class is not thread-safe on multi-thread applications. If a pointer to an instance of this class will be passed to multiple threads, the code in those threads must guarantee thread-safe usage of the class.

File Stem

tsetitr

Base

somf_TIterator

Metaclass

SOMClass

Ancestor Classes

somf_TIterator, SOMObject

New Methods

somfTSetIteratorInit

Overriding Methods

somUninit
somfNext
somfFirst
somfRemove

somfFirst Method

Purpose

Resets the iterator and returns the first element of a set.

IDL Syntax

somf_MCollectible somfFirst ( );

Description

The **somfFirst** method resets the iterator and returns the first element of the set that corresponds to the set iterator represented by the receiving object.

The **somfFirst** method resets the iterator to the beginning of the set. This is true not only the first time the iterator is used; it is also true if other operations on the collection cause the iterator to be invalidated. In the second case, the method also revalidates the iterator.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. **somfFirst** is a method name declared in multiple parents (for example: **somf_TSequence**, **somf_TIterator**, etc.). You will probably have to fully qualify the method name (for example: **somf_TSetIterator_somfFirst**). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you could have referenced this method as:

```
itr->somfFirst(ev);
```

Parameters

receiver	A pointer to an object of class **somf_TSetIterator.**
ev	A pointer to the **Environment** structure for the calling method.

Return Value

This method returns a pointer to the first **somf_MCollectible** object in the set. Or, SOMF_NIL is returned if the collection is empty.

Example

```
somf_TSet s;
Environment *ev;
somf_TSetIterator itr;
somf_MCollectible itrobj;

ev = somGetGlobalEnvironment();

s = somf_TSetNew();
itr = somf_TSetIteratorNew();
_somfTSetIteratorInit(itr, ev, s);

/* Add some object to s */

/* Iterate through the TSet */
itrobj = somf_TSetIterator_somfFirst(itr,ev);
while (itrobj != SOMF_NIL)
{
    /* Do something with itrobj */
    itrobj = _somfNext(itr,ev);
}

_somFree (s);
_somFree (itr);
```

Original Class

somf_TIterator

Related Information

Methods: somfNext

somfNext Method

Purpose

Gets the next object in a set.

IDL Syntax

somf_MCollectible somfNext ( );

Description

The **somfNext** method determines the next object in the set that corresponds to the set iterator represented by the receiving object, and returns a pointer to it. Objects are retrieved in an order reflecting the “ordered–ness” of the set (or the lack of ordering on the set elements).

If the **somf_TSet** collection has changed (other than through the use of the **somfRemove** method of this iterator) since the last time the **somfFirst** method was called, the iterator becomes invalid and will *fail* if asked to find the next object. For example, if the collection’s **somfAdd** method were called after starting to iterate through the collection, the iterator then would not allow iteration to continue. The iterator must be reset, and the easiest way to do that is to call the iterator’s **somfFirst** method and start over.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Thus, if any child of **somf_TIterator** is used with a child of **somf_TPrimitiveLinkedListIterator**, then the name of the method will have to be fully qualified (example: **somf_TSetIterator_somfNext**). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you could have referenced this method as:

```
itr->somfNext(ev);
```

Parameters

receiver	A pointer to an object of class **somf_TSetIterator.**
ev	A pointer to the **Environment** structure for the calling method.

Return Value

There are two possible valid return values for this method:

somf_MCollectible
: A pointer to the next **somf_MCollectible** object in the set.

SOMF_NIL
: The end of the set has been reached.

Example

```
somf_TSet s;
Environment *ev;
somf_TSetIterator itr;
somf_MCollectible itrobj;

ev = somGetGlobalEnvironment();

s = somf_TSetNew();
itr = somf_TSetIteratorNew();
_somfTSetIteratorInit(itr, ev, s);

/* Add some object to s */
```

```
/* Iterate through the TSet */
itrobj = somf_TSetIterator_somfFirst(itr,ev);
while (itrobj != SOMF_NIL)
{
    /* Do something with itrobj */
    itrobj = _somfNext(itr,ev);
}

_somFree (s);
_somFree (itr);
```

Original Class

somf_TIterator

Related Information

Methods: somfFirst

somfRemove Method

Purpose

Removes the current object (the one just returned by **somfFirst** or **somfNext**) from a set.

IDL Syntax

void somfRemove ( );

Description

The **somfRemove** method removes the current object (the one just returned by **somfFirst** or **somfNext**) from the set that corresponds to the iterator used as the receiving object.

This method is the only way to remove an object from a set during iteration. However, if multiple iterators are in process, all other iterators are invalidated, just as if some other kind of change had occurred in the collection.

If the collection has changed since the last time **somfFirst** was called (other than through the use of the **somfRemove** method of this iterator), this method will *fail*.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. **somfRemove** is a method name declared in multiple parents (for example: **somf_TCollection**, **somf_THashTable**, **somf_TIterator**, etc.) You will probably have to fully qualify the method name (for example: **somf_TSetIterator_somfRemove**). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you could have referenced this method as:

```
itr->somfRemove(ev);
```

Parameters

receiver	A pointer to an object of class **somf_TSetIterator.**
ev	A pointer to the **Environment** structure for the calling method.

Return Value

None.

Example

```
somf_TSet s;
Environment *ev;
somf_TSetIterator itr;

ev = somGetGlobalEnvironment();

s = somf_TSetNew();
itr = somf_TSetIteratorNew();
_somfTSetIteratorInit(itr, ev, s);

/* Remove the first object in s */

/* Iterate through the TSet */
somf_TSetIterator_somfFirst(itr,ev);
somf_TSetIterator_somfRemove(itr,ev);

_somFree (s);
_somFree (itr);
```

Original Class

somf_TIterator

somfTSetIteratorInit Method

Purpose

Initializes **somf_TSetIterator** object, establishing it as the iterator for a given **somf_TSet** set.

IDL Syntax

somf_TSetIterator somfTSetIteratorInit (in somf_TSet *h*);

Description

The **somfTSetIteratorInit** method initializes a given iterator object (the **somf_TSetIterator** receiving object) that will iterate over the specified **somf_TSet** set.

Note: This is one of two ways to initialize a **somf_TSetIterator** to point to an instance of a **somf_TSet** set. The other way is to use the **somf_TSet** class's **somfCreateIterator** method described on page 271.

Note: You cannot override this method.

Parameters

receiver	A pointer to an object of class **somf_TSetIterator.**
ev	A pointer to the **Environment** structure for the calling method.
h	A pointer to the set that the receiving object will iterate over.

Return Value

This method returns a pointer to an initialized **somf_TSetIterator** object.

Example

```
somf_TSet s;
Environment *ev;
somf_TSetIterator itr;

ev = somGetGlobalEnvironment();

s = somf_TSetNew();
itr = somf_TSetIteratorNew();
_somfTSetIteratorInit(itr, ev, s);

_somFree (s);
_somFree (itr);
```

Original Class

somf_TSetIterator

somf_TSortedSequence Class

Description

The **somf_TSortedSequence** class is a child of the **somf_TSequence** class. Ordering of objects in a sorted sequence collection is based on how the objects relate to each other, ranging from largest to smallest. Any object in the **somf_TSortedSequence** "IsGreaterThan" or "IsEqualTo" the object behind it, and "IsLessThan" or "IsEqualTo" the element in front of it.

When you link, include the following library reference to get access to this class: **somtk**

Warning: Do not be misled by the interface of methods in this class, many of which are overridden from the **somf_TSequence** or **somf_TCollection** class. All objects placed into a **somf_TSortedSequence** collection *must* be instances of the **somf_MOrderableCollectible** class. If you attempt to add a **somf_MCollectible** object to a sorted sequence, the class method will abend.

All **somf_MOrderableCollectible** objects that are inserted into a **somf_TSortedSequence** collection should override the **somfIsEqual**, **somfIsLessThan**, **somfIsGreaterThan**, and **somfHash** methods.

The **somf_TSortedSequence** class uses **somfIsEqual** to compare objects in the collection. You *cannot* override or change this to the **somfIsSame** method.

Although the methods in this class are reentrant, the class is not thread-safe on multi-thread applications. If a pointer to an instance of this class will be passed to multiple threads, the code in those threads must guarantee thread-safe usage of the class.

File Stem

tss

Base

somf_TSequence

Metaclass

SOMClass

Ancestor Classes

somf_TSequence, somf_TCollection, somf_MCollectible, SOMObject

New Methods

somfCreateSequenceIterator
somfGetSequencingFunction
somfSetSequencingFunction
somfCreateSortedSequenceNode
somfAssign
somfTSortedSequenceInitF
somfTSortedSequenceInitS

Overriding Methods

somInit
somUninit
somfAdd

somfRemove
somfDeleteAll
somfRemoveAll
somfCount
somfAfter
somfBefore
somfLast
somfFirst
somfMember
somfCreateIterator
somfOccurrencesOf

somfAdd Method

Purpose

Adds an `obj` to a sorted sequence.

IDL Syntax

somf_MCollectible somfAdd (in somf_MCollectible *obj*);

Description

The **somfAdd** method adds an object *obj* to the sorted sequence represented by the receiving object.

Notice that the **somfAdd** method does not include an argument specifying where to add the object, because the sequence will be ordered based on how the elements relate to each other.

Parameters

receiver	A pointer to an object of class **somf_TSortedSequence**
ev	A pointer to the **Environment** structure for the calling method.
obj	A pointer to an **somf_MCollectible** that will be added to this instance.

Return Value

This method returns a pointer to the **somf_MCollectible** object added.

Example

```
somf_TSortedSequence ss;
<Your class which inherits from somf_MOrderableCollectible> obj;
Environment *ev;

ev = somGetGlobalEnvironment();

ss = somf_TSortedSequenceNew();
obj =
   <Your class which inherits from somf_MOrderableCollectible>New();

/* Add obj to ss */
_somfAdd(ss, ev, obj);

_somFree (ss);
```

Original Class

somf_TCollection (overridden here)

somfAfter Method

Purpose

Gets the object found after a given object in a sorted sequence.

IDL Syntax

somf_MCollectible somfAfter (in somf_MCollectible *obj***);**

Description

The **somfAfter** method returns the object found after the specified object *obj* in the sorted sequence represented by the receiving object.

Parameters

receiver	A pointer to an object of class **somf_TSortedSequence.**
ev	A pointer to the **Environment** structure for the calling method.
obj	A pointer to the **somf_MCollectible** object that precedes the returned obj.

Return Value

There are two possible valid return values for this method:

somf_MCollectible
A pointer to the **somf_MCollectible** object after *obj*.

SOMF_NIL The designated *obj* is the last object in this collection or was not found.

Example

```
somf_TSortedSequence ss;
<Your class which inherits from somf_MOrderableCollectible> obj;
<Your class which inherits from somf_MOrderableCollectible> objptr;
Environment *ev;

ev = somGetGlobalEnvironment();

ss = somf_TSortedSequenceNew();
obj = <Your class which inherits from
                                  somf_MOrderableCollectible>New();

/* Determine what object comes after obj */
objptr =
  (<Your class which inherits from somf_MOrderableCollectible>*)
   _somfAfter(ss,ev,obj);

_somFree (ss);
```

Original Class

somf_TSequence (overridden here)

Related Information

Methods: somfBefore, somfFirst, somfLast

somfAssign Method

Purpose

Assigns a sorted sequence as equal to a given source sorted sequence.

IDL Syntax

void somfAssign (in somf_TSortedSequence *s*);

Description

The **somfAssign** method assigns the sorted sequence represented by the receiving object as equal to the specified source sorted sequence. That is, the method sets/resets the instance variables of the receiver to the values of the source. This operation is logically equivalent to using the "=" operator.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Thus, if any child of **somf_TSortedSequence** is used with any other main collection class, then the name of the method will have to be fully qualified (for example: **somf_TSortedSequence_somfAssign**). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you could have referenced this method as:

```
d->somfAssign(ev, d2);
```

Parameters

receiver	A pointer to an object of class **somf_TSortedSequence.**
ev	A pointer to the **Environment** structure for the calling method.
s	A pointer to the sorted sequence to which the receiver will be equal.

Return Value

None.

Example

```
somf_TSortedSequence s1;
somf_TSortedSequence s2;
Environment *ev;

ev = somGetGlobalEnvironment();

s1 = somf_TSortedSequenceNew();
s2 = somf_TSortedSequenceNew();

/* Add som objects to s1 */

/* Assign s2 = s1 */
somf_TSortedSequence_somfAssign(s2, ev, s1);

_somFree (s1);
_somFree (s2);
```

Original Class

somf_TSortedSequence

somfBefore Method

Purpose

Returns the object found before a given object in a sorted sequence.

IDL Syntax

somf_MCollectible somfBefore (in somf_MCollectible *obj***);**

Description

The **somfBefore** method returns the object found before the specified object *obj* in the sorted sequence represented by the receiving object.

Parameters

receiver	A pointer to an object of class **somf_TSortedSequence.**
ev	A pointer to the **Environment** structure for the calling method.
obj	A pointer to the **somf_MCollectible** object that is behind the returned obj.

Return Value

There are two possible valid return values for this method:

somf_MCollectible
A pointer to the **somf_MCollectible** object that precedes *obj*.

SOMF_NIL The designated *obj* is the first object in the sorted sequence or was not found.

Example

```
somf_TSortedSequence ss;
<Your class which inherits from somf_MOrderableCollectible> obj;
<Your class which inherits from somf_MOrderableCollectible> objptr;
Environment *ev;

ev = somGetGlobalEnvironment();

ss = somf_TSortedSequenceNew();
obj =
  <Your class which inherits from somf_MOrderableCollectible>New();

/* Determine what object comes before obj */
objptr =
  (<Your class which inherits from somf_MOrderableCollectible>*)
   _somfBefore(ss,ev,obj);

_somFree (ss);
```

Original Class

somf_TSequence (overridden here)

Related Information

Methods: somfAfter, somfFirst, somfLast

somfCount Method

Purpose

Gets the number of objects in a given sorted sequence.

IDL Syntax

long somfCount ( );

Description

The **somfCount** method determines the number of objects in the sorted sequence given as the receiving object, and returns that number.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Thus, if any child of **somf_TCollection** is used with a child of **somf_THashTable**, then the name of the method will have to be fully qualified (for example: **somf_TDictionary_somfCount**). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you could have referenced this method as:

```
d->somfCount(ev);
```

Parameters

receiver	A pointer to an object of class **somf_TSortedSequence.**
ev	A pointer to the **Environment** structure for the calling method.

Return Value

This method returns the number of objects in this sorted sequence.

Example

```
somf_TSortedSequence ss;
Environment *ev;

ev = somGetGlobalEnvironment();

ss = somf_TSortedSequenceNew();

somPrintf("\n Count of ss= %d\n", _somfCount(ss,ev));

_somFree (ss);
```

Original Class

somf_TCollection (overridden here)

somfCreateIterator Method

Purpose

Returns a new iterator that is suitable for iterating over the objects in a sorted sequence.

IDL Syntax

somf_TIterator somfCreateIterator ( );

Description

The **somfCreateIterator** method returns a new iterator that is suitable for iterating over the objects in the sorted sequence given as the receiving object.

Note: This is one of three ways to initialize a **somf_TSortedSequenceIterator** to point to an instance of a **somf_TSortedSequence**. One other way is to use the initializer method of the **somf_TSortedSequenceIterator** class, described on page 334. The final way is to use the **somf_TSortedSequence** class's **somfCreateSequenceIterator** method described on page 308.

Parameters

receiver A pointer to an object of class **somf_TSortedSequence.**

ev A pointer to the **Environment** structure for the calling method.

Return Value

This method returns a pointer to the new iterator.

Example

```
somf_TSortedSequence ss;
Environment *ev;
somf_TSortedSequenceIterator itr;

ev = somGetGlobalEnvironment();

ss = somf_TSortedSequenceNew();
itr = (somf_TSortedSequenceIterator*) _somfCreateIterator(ss,ev);

_somFree (ss);
_somFree (itr);
```

Original Class

somf_TCollection (overridden here)

Related Information

Methods: somfCreateSequenceIterator

somfCreateSequenceIterator Method

Purpose

Returns a new iterator that is suitable for iterating over the objects in a sorted sequence.

IDL Syntax

somf_TSequenceIterator somfCreateSequenceIterator ( );

Description

The **somfCreateSequenceIterator** method returns a new iterator that is suitable for iterating over the objects in the sorted sequence represented by the receiving object.

Note: This is one of three ways to initialize a **somf_TSortedSequenceIterator** to point to an instance of a **somf_TSortedSequence**. One other way is to use the initializer method for the **somf_TSortedSequenceIterator** class, described on page 334. The final way is to use the **somf_TSortedSequence** class's **somfCreateIterator** method described on page 307.

This method is virtually identical to the **somfCreateIterator** method; thus, you could use either one. The only difference between methods is the indicated type of their return value: the current method returns a **somf_TSequenceIterator** object, whereas the **somfCreateIterator** method returns a **somf_TIterator** object. In reality, however, both methods return an instance of a **somf_TSortedSequenceIterator** that has been typed correctly.

Parameters

receiver A pointer to an object of class **somf_TSortedSequence.**

ev A pointer to the **Environment** structure for the calling method.

Return Value

This method returns a pointer to the new sorted-sequence iterator.

Example

```
somf_TSortedSequence ss;
Environment *ev;
somf_TSortedSequenceIterator itr;

ev = somGetGlobalEnvironment();

ss = somf_TSortedSequenceNew();
itr = (somf_TSortedSequenceIterator*)
                              _somfCreateSequenceIterator(ss,ev);

_somFree (ss);
_somFree (itr);
```

Original Class

somf_TSortedSequence

Related Information

Methods: somfCreateIterator

somfCreateSortedSequenceNode Method

Purpose

Creates a new **somf_TSortedSequenceNode** in a **somf_TSortedSequence** collection, given a key to the new node and its left and right children.

IDL Syntax

```
somf_TSortedSequenceNode  somfCreateSortedSequenceNode (
                                    in somf_TSortedSequenceNode n1,
                                    in somf_MOrderableCollectible obj,
                                    in somf_TSortedSequenceNode n2);
```

Description

The **somfCreateSortedSequenceNode** method creates a new node of class **somf_TSortedSequenceNode** in the **somf_TSortedSequence** collection represented by the receiving object. The method call also specifies a **somf_MOrderableCollectible** object to be used as the key to the new node, as well as two **somf_TSortedSequenceNode** objects to be used as the left and right children of the new node.

If you create a new class that inherits from the **somf_TSortedSequence** class, you might consider overriding this method in order to customize how an instance of your new class creates a new node.

Parameters

receiver	A pointer to an object of class **somf_TSortedSequence.**
ev	A pointer to the **Environment** structure for the calling method.
n1	A pointer to the left child of the new **somf_TSortedSequenceNode** object.
obj	A pointer to the key of the new **somf_TSortedSequenceNode** object.
n2	A pointer to the right child of the new **somf_TSortedSequenceNode** object.

Return Value

This method returns a pointer to the new **somf_TSortedSequenceNode** created.

Original Class

somf_TSortedSequence

somfDeleteAll Method

Purpose

Removes all objects from a sorted sequence and deallocates the storage that these objects might have owned. (That is, the destructor function is called for each object in the collection.)

IDL Syntax

void somfDeleteAll ( );

Description

The **somfDeleteAll** method removes all of the objects from the sorted sequence represented by the receiving object. The method also deallocates the storage that these objects might have owned (that is, the destructor function is called for each object in the collection).

Be careful with **somfDeleteAll**. Since a collection only contains *pointers* to objects (rather than the objects themselves), **somfDeleteAll** can cause a problem if a pointer to an object appears more than once. For example, if multiple pointers to 'A' exists, or if a single pointer to 'A' is in the collection multiple times, the behavior of the code is undefined, because it will try to delete 'A' multiple times. If you think there is a chance that an object could appear in the collection more than once, you should consider using **somfRemoveAll** to remove the objects from the collection and deleting them some other way.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Thus, if any child of **somf_TCollection** is used with **somf_THashTable**, then the name of the method will have to be fully qualified (for example: **somf_TDictionary_somfDeleteAll**). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you could have referenced this method as:

```
d->somfDeleteAll(ev);
```

Parameters

receiver	A pointer to an object of class **somf_TSortedSequence.**
ev	A pointer to the **Environment** structure for the calling method.

Return Value

None.

Example

```
somf_TSortedSequence ss;
Environment *ev;

ev = somGetGlobalEnvironment();

ss = somf_TSortedSequenceNew();

/* Remove all the objects from ss AND DELETE THEM */
somf_TSortedSequence_somfDeleteAll(ss,ev);

_somFree (ss);
```

Original Class

somf_TCollection (overridden here)

somfFirst Method

Purpose

Gets the first object in a sorted sequence.

IDL Syntax

somf_MCollectible somfFirst ( );

Description

The **somfFirst** method determines the first object in the sorted sequence represented by the receiving object, and returns a pointer to the object.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. **somfFirst** is a method name declared in multiple parents (for example: **somf_TSequence**, **somf_TIterator**, etc.). You will probably have to fully qualify the method name (for example: **somf_TSortedSequence_somfFirst**). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you could have referenced this method as:

```
seq->somfFirst(ev);
```

Parameters

receiver A pointer to an object of class **somf_TSortedSequence.**

ev A pointer to the **Environment** structure for the calling method.

Return Value

There are two possible valid return values for this method:

somf_MCollectible
A pointer to the first **somf_MCollectible** object in the sorted sequence.

SOMF_NIL Nothing is in the collection.

Example

```
somf_TSortedSequence ss;
Environment *ev;
somf_MOrderableCollectible obj;

ev = somGetGlobalEnvironment();

ss = somf_TSortedSequenceNew();


/* Determine the first object in ss */
obj = somf_TSortedSequence_somfFirst(ss,ev);

_somFree (ss);
```

Original Class

somf_TSequence (overridden here)

Related Information

Methods: somfLast, somfAfter, somfBefore

somfGetSequencingFunction Method

Purpose

Gets a pointer to the function used to compare objects in a sorted sequence, and consequently determines the sequence of the collection.

IDL Syntax

somf_MBetterOrderableCompareFn somfGetSequencingFunction ( );

Description

The **somfGetSequencingFunction** method returns a pointer to the function used to compare objects in the sorted sequence represented by the receiving object. This consequently reveals the sequence of the collection.

Parameters

receiver A pointer to an object of class **somf_TSortedSequence.**

ev A pointer to the **Environment** structure for the calling method.

Return Value

This method returns a pointer to the compare method used by this **somf_TSortedSequence** object.

Example

```
somf_TSortedSequence ss;
Environment *ev;

ev = somGetGlobalEnvironment();

ss = somf_TSortedSequenceNew();

if (_somfGetSequencingFunction(ss,ev) !=
      somf_MOrderableCollectibleClassData.somfCompare)
{
    somPrintf("\n What Compare Function are we using?\n");
}

_somFree (ss);
```

Original Class

somf_TSortedSequence

Related Information

Methods: somfSetSequencingFunction

somfLast Method

Purpose

Gets the last object in a sorted sequence.

IDL Syntax

somf_MCollectible somfLast ( );

Description

The **somfLast** method determines the last object in the sorted sequence represented by the receiving object, and returns a pointer to it.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. **somfLast** is a method name declared in multiple parents (for example: **somf_TSequenceIterator**, **somf_TSequence**, etc.). You will probably have to fully qualify the method name (for example: **somf_TSortedSequence_somfLast**). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you could have referenced this method as:

```
seq->somfLast(ev);
```

Parameters

receiver A pointer to an object of class **somf_TSortedSequence.**

ev A pointer to the **Environment** structure for the calling method.

Return Value

There are two possible valid return values for this method:

somf_MCollectible
A pointer to the last **somf_MCollectible** object in the collection.

SOMF_NIL Nothing is in the collection.

Example

```
somf_TSortedSequence ss;
Environment *ev;
somf_MOrderableCollectible obj;

ev = somGetGlobalEnvironment();

ss = somf_TSortedSequenceNew();

/* Determine the last object in ss */
obj = somf_TSortedSequence_somfLast(ss,ev);

_somFree (ss);
```

Original Class

somf_TSequence (overridden here)

Related Information

Methods: somfFirst, somfAfter, somfBefore

somfMember Method

Purpose

Gets an object in a sorted sequence.

IDL Syntax

somf_MCollectible somfMember (in somf_MCollectible *obj*);

Description

The **somfMember** method determines whether a specified object *obj* is in the sorted sequence represented by the receiving object and, if found, returns a pointer to it.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Thus, if any child of **somf_TCollection** is used with **somf_THashTable**, then the name of the method will have to be fully qualified (for example: **somf_TSortedSequence_somfMember**). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you could have referenced this method as:

```
d->somfMember(ev, obj);
```

Parameters

receiver A pointer to an object of class **somf_TSortedSequence.**

ev A pointer to the **Environment** structure for the calling method.

obj A pointer to the **somf_MCollectible** object that may or may not be a member of the collection.

Return Value

There are two possible valid return values for this method:

somf_MCollectible
A pointer to the object the method determined as a member of the collection.

SOMF_NIL The object was not found.

Example

```
somf_TSortedSequence ss;
<your Class which inherits from somf_MOrderableCollectible> obj;
Environment *ev;

ev = somGetGlobalEnvironment();

ss = somf_TSortedSequenceNew();
obj =
   <your Class which inherits from somf_MOrderableCollectible>New();

_somfAdd(ss, ev, obj);

if (_somfMember(ss, ev, obj) != SOMF_NIL)
    somPrintf("\n obj is a Member\n");
else
    somPrintf("\n ERROR: obj should be a Member\n");

_somFree (ss);
_somFree (obj);
```

Original Class

somf_TCollection (overridden here)

Related Information

Methods: somfOccurrencesOf

somfOccurrencesOf Method

Purpose

Determines the number of times an object is in a sorted sequence.

IDL Syntax

long somfOccurrencesOf (in somf_MCollectible *obj***);**

Description

The **somfOccurrencesOf** method determines the number of times an object *obj* is in the sorted sequence represented by the receiving object, and returns that number.

Parameters

receiver	A pointer to an object of class **somf_TSortedSequence**
ev	A pointer to the **Environment** structure for the calling method.
obj	A pointer to the **somf_MCollectible** to look for in the collection.

Return Value

This method returns the number of times *obj* occurs in the sorted sequence.

Example

```
somf_TSortedSequence ss;
<your Class which inherits from somf_MOrderableCollectible> obj;
Environment *ev;

ev = somGetGlobalEnvironment();

ss = somf_TSortedSequenceNew();
obj =
   <your Class which inherits from somf_MOrderableCollectible>New();

somPrintf("\n There are %d OccurrencesOf obj\n",
           _somfOccurrencesOf(ss, ev, obj));

_somFree (ss);
_somFree (obj);
```

Original Class

somf_TSequence (overridden here)

Related Information

Methods: somfMember

somfRemove Method

Purpose

Removes an object from a sorted sequence.

IDL Syntax

somf_MCollectible somfRemove (in somf_MCollectible *obj***);**

Description

The **somfRemove** method removes the specified object *obj* from the sorted sequence represented by the receiving object.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. **somfRemove** is a method name declared in multiple parents (for example: **somf_TCollection**, **somf_THashTable**, **somf_TIterator**, etc.) You will probably have to fully qualify the method name (as **somf_TSortedSequence_somfRemove,** for example). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you could have referenced this method as:

```
d->somfRemove(ev, obj);
```

Parameters

receiver	A pointer to an object of class **somf_TSortedSequence.**
ev	A pointer to the **Environment** structure for the calling method.
obj	A pointer to the **somf_MCollectible** object to be removed from the collection.

Return Value

There are two possible valid return values for this method:

somf_MCollectible
: A pointer to the object that was actually removed.

SOMF_NIL
: The specified object was not found.

Example

```
somf_TSortedSequence ss;
<your Class which inherits from somf_MOrderableCollectible> obj;
Environment *ev;

ev = somGetGlobalEnvironment();

ss = somf_TSortedSequenceNew();
obj =
   <your Class which inherits from somf_MOrderableCollectible>New();

/* Remove obj from ss */
somf_TSortedSequence_somfRemove(ss,ev,obj);

_somFree (ss);
_somFree (obj);
```

Original Class

somf_TCollection (overridden here)

Related Information

Methods: somfRemoveAll

somfRemoveAll Method

Purpose

Removes all of the objects from a sorted sequence.

IDL Syntax

void somfRemoveAll ( );

Description

The **somfRemoveAll** method removes all of the objects from the sorted sequence represented by the receiving object.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Thus, if any child of **somf_TCollection** is used with **somf_THashTable**, then the name of the method will have to be fully qualified (for example: **somf_TSortedSequence_somfRemoveAll**). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you could have referenced this method as:

```
d->somfRemoveAll(ev);
```

Parameters

receiver	A pointer to an object of class **somf_TSortedSequence.**
ev	A pointer to the **Environment** structure for the calling method.

Return Value

None.

Example

```
somf_TSortedSequence ss;
Environment *ev;

ev = somGetGlobalEnvironment();

ss = somf_TSortedSequenceNew();

/* Remove all the objects from ss */
somf_TSortedSequence_somfRemoveAll(ss,ev);

_somFree (ss);
```

Original Class

somf_TCollection (overridden here)

Related Information

Methods: somfRemove

somfSetSequencingFunction Method

Purpose

Sets a pointer to the method used to compare objects in a sorted sequence, and consequently determines the sequence of the collection.

IDL Syntax

void somfSetSequencingFunction (in somf_MBetterOrderableCompareFn *fn*);

Description

The **somfSetSequencingFunction** method sets a pointer to the method that will be used to compare objects in the sorted sequence represented by the receiving object. This consequently determines the sequence of the collection.

Parameters

receiver A pointer to an object of class **somf_TSortedSequence.**

ev A pointer to the **Environment** structure for the calling method.

fn A pointer to the compare method to be used by this **somf_TSortedSequence** object.

This should always be set to:

```
somf_MOrderableCollectibleClassData.somfCompare.
```

This specification is necessary because SOM needs a pointer to the original declaration of the method, which resides in **somf_MOrderableCollectible**. The **somf_TSortedSequence** object will use this pointer to access the **somfCompare** method that was declared and defined in the object being inserted into, or removed from, the **somf_TSortedSequence** object.

Return Value

None.

Example

```
somf_TSortedSequence ss;
Environment *ev;

ev = somGetGlobalEnvironment();

ss = somf_TSortedSequenceNew();

_somfSetSequencingFunction(ss, ev,
                  somf_MOrderableCollectibleClassData.somfCompare);

_somFree (ss);
```

Original Class

somf_TSortedSequence

Related Information

Methods: somfGetSequencingFunction

somfTSortedSequenceInitF Method

Purpose

Initializes a new sorted sequence, given the comparison method that it will use.

IDL Syntax

somf_TSortedSequence somfTSortedSequenceInitF(
in somf_MBetterOrderableCompareFn *testfn***);**

Description

The **somfTSortedSequenceInitF** method initializes the new sorted sequence represented by the receiving object, given a pointer to the compare method that the new object will use.

Note: You cannot override this method.

Parameters

receiver A pointer to an object of class **somf_TSortedSequence.**

ev A pointer to the **Environment** structure for the calling method.

testfn A pointer to the compare method to be used by this **somf_TSortedSequence** object.

This should always be set to:

```
somf_MOrderableCollectibleClassData.somfCompare.
```

This specification is necessary because SOM needs a pointer to the original declaration of the method, which resides in **somf_MOrderableCollectible**. The **somf_TSortedSequence** object will use this pointer to access the **somfCompare** method that was declared and defined in the object being inserted into, or removed from, the **somf_TSortedSequence** object.

Return Value

This method returns a pointer to an initialized **somf_TSortedSequence** object.

Example

```
somf_TSortedSequence s1;
Environment *ev;

ev = somGetGlobalEnvironment();

s1 = somf_TSortedSequenceNew();
_somfTSortedSequenceInitF(s1, ev,
                  somf_MOrderableCollectibleClassData.somfCompare);

_somFree (s1);
```

Original Class

somf_TSortedSequence

Related Information

Methods: somfTSortedSequenceInitS

somfTSortedSequenceInitS Method

Purpose

Initializes a new sorted sequence, setting it equal to another given sorted sequence.

IDL Syntax

somf_TSortedSequence somfTSortedSequenceInitS (in somf_TSortedSequence *s*);

Description

The **somfTSortedSequenceInitS** method initializes the new sorted sequence represented by the receiving object. The method sets the new sorted sequence equal to a specified source sorted sequence.

Note: You cannot override this method.

Parameters

receiver	A pointer to an object of class **somf_TSortedSequence.**
ev	A pointer to the **Environment** structure for the calling method.
s	A pointer to the **somf_TSortedSequence** object to which the new sorted sequence will be equal.

Return Value

This method returns a pointer to an initialized **somf_TSortedSequence** object.

Example

```
somf_TSortedSequence s1;
somf_TSortedSequence s2;
Environment *ev;

ev = somGetGlobalEnvironment();

s1 = somf_TSortedSequenceNew();
s2 = somf_TSortedSequenceNew();
_somfTSortedSequenceInitS(s2, ev, s1);

_somFree (s1);
_somFree (s2);
```

Original Class

somf_TSortedSequence

Related Information

Methods: somfTSortedSequenceInitF

somf_TSortedSequenceIterator Class

Description

This class defines an iterator for the **somf_TSortedSequence** class that will iterate over all of the objects in a sorted sequence.

When you link, include the following library reference to get access to this class: **somtk**

Although the methods in this class are reentrant, the class is not thread-safe on multi-thread applications. If a pointer to an instance of this class will be passed to multiple threads, the code in those threads must guarantee thread-safe usage of the class.

File Stem

tssitr

Base

somf_TSequenceIterator

Metaclass

SOMClass

Ancestor Classes

somf_TSequenceIterator, somf_TIterator, SOMObject

New Methods

somfStartHere
somfTSortedSequenceIteratorInit

Overriding Methods

somfFirst
somfNext
somfLast
somfPrevious
somfRemove

somfFirst Method

Purpose

Resets the iterator and returns the first object of a sorted sequence.

IDL Syntax

somf_MCollectible somfFirst ( );

Description

The **somfFirst** method resets the **somf_TSortedSequenceIterator** iterator given as the receiving object. The method also returns the first object of the **somf_TSortedSequence** collection that corresponds to the specified iterator.

This method resets the iterator to the beginning of the sorted sequence. This is true not only the first time the iterator is used; it is also true if other operations on the collection cause the iterator to be invalidated. In the second case, this method also revalidates the iterator.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. **somfFirst** is a method name declared in multiple parents (for example: **somf_TSequence**, **somf_TIterator**, etc.). You will probably have to fully qualify the method name (for example: **somf_TSortedSequenceIterator_somfFirst**). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you could have referenced this method as:

```
itr->somfFirst(ev);
```

Parameters

receiver	A pointer to an object of class **somf_TSortedSequenceIterator.**
ev	A pointer to the **Environment** structure for the calling method.

Return Value

This method returns a pointer to the first **somf_MCollectible** object in the sorted sequence. Or, SOMF_NIL is returned if the collection is empty.

Example

```
somf_TSortedSequence ss;
Environment *ev;
somf_TSortedSequenceIterator itr;
<Your class which inherits from somf_MOrderableCollectible> itrobj;

ev = somGetGlobalEnvironment();

ss = somf_TSortedSequenceNew();
itr = somf_TSortedSequenceIteratorNew();
_somfTSortedSequenceIteratorInit(itr, ev, ss);

/* Iterate through the TSortedSequence */
itrobj =
      (<Your class which inherits from somf_MOrderableCollectible>*)
      somf_TSortedSequenceIterator_somfFirst(itr,ev);
while (itrobj != SOMF_NIL)
{
    /* Do something with itrobj */
```

```
        itrobj =
          (<Your class which inherits from somf_MOrderableCollectible>*)
          _somfNext(itr,ev);
    }

    _somFree (ss);
    _somFree (itr);
```

Original Class

somf_TIterator (overridden here)

Related Information

Methods: somfNext, somfStartHere

somfLast Method

Purpose

Gets the last object in a sorted sequence.

IDL Syntax

somf_MCollectible somfLast ( );

Description

The **somfLast** method determines the last object in the **somf_TSortedSequence** collection that corresponds to the **somf_TSortedSequenceIterator** iterator used as the receiving object.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. **somfLast** is a method name declared in multiple parents (for example: **somf_TSequenceIterator**, **somf_TSequence**, etc.). You will probably have to fully qualify the method name (for example: **somf_TSortedSequenceIterator_somfLast**). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you could have referenced this method as:

```
itr->somfLast(ev);
```

Parameters

receiver	A pointer to an object of class **somf_TSortedSequenceIterator.**
ev	A pointer to the **Environment** structure for the calling method.

Return Value

This method returns a pointer to the last **somf_MCollectible** object in the sorted sequence. Or, SOMF_NIL is returned if the collection is empty.

Example

```
somf_TSortedSequence ss;
Environment *ev;
somf_TSortedSequenceIterator itr;
<Your class which inherits from somf_MOrderableCollectible> itrobj;

ev = somGetGlobalEnvironment();

ss = somf_TSortedSequenceNew();
itr = somf_TSortedSequenceIteratorNew();
_somfTSortedSequenceIteratorInit(itr, ev, ss);


/* Go to the last object in ss */
itrobj =
      (<Your class which inherits from somf_MOrderableCollectible>*)
      somf_TSortedSequenceIterator_somfLast(itr,ev);

_somFree (ss);
_somFree (itr);
```

Original Class

somf_TSequenceIterator (overridden here)

Related Information

Methods: somfPrevious

somfNext Method

Purpose

Gets the next object in a sorted sequence.

IDL Syntax

somf_MCollectible somfNext ( );

Description

The **somfNext** method determines the next object in the **somf_TSortedSequence** collection that corresponds to the **somf_TSortedSequenceIterator** iterator used as the receiving object, and returns a pointer to it. Objects are retrieved in an order that reflects the "ordered–ness" of the sorted sequence (or the lack of ordering on the sorted sequence objects).

If the **somf_TSortedSequence** collection has changed (other than through the use of the **somfRemove** method of this iterator) since the last time the **somfFirst** method was called, the iterator becomes invalid and will *fail* if asked to find the next object. For example, if the collection's **somfAdd** method were called after starting to iterate through the collection, the iterator then would not allow iteration to continue. The iterator must be reset, and the easiest way to do that is to call the iterator's **somfFirst** method and start over.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Thus, if any child of **somf_TIterator** is used with a child of **somf_TPrimitiveLinkedListIterator**, then the name of the method will have to be fully qualified (for example: **somf_TSortedSequenceIterator_somfNext**). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you could have referenced this method as:

```
itr->somfNext(ev);
```

Parameters

receiver A pointer to an object of class **somf_TSortedSequenceIterator.**

ev A pointer to the **Environment** structure for the calling method.

Return Value

There are two possible valid return values for this method:

somf_MCollectible
A pointer to the next **somf_MCollectible** object in the sorted sequence.

SOMF_NIL The end of the collection has been reached.

Example

```
somf_TSortedSequence ss;
Environment *ev;
somf_TSortedSequenceIterator itr;
<Your class which inherits from somf_MOrderableCollectible> itrobj;

ev = somGetGlobalEnvironment();

ss = somf_TSortedSequenceNew();
itr = somf_TSortedSequenceIteratorNew();
_somfTSortedSequenceIteratorInit(itr, ev, ss);
```

```
/* Iterate through the TSortedSequence */
itrobj =
      (<Your class which inherits from somf_MOrderableCollectible>*)
      somf_TSortedSequenceIterator_somfFirst(itr,ev);
while (itrobj != SOMF_NIL)
{
    /* Do something with itrobj */

    itrobj =
      (<Your class which inherits from somf_MOrderableCollectible>*)
      _somfNext(itr,ev);
}

_somFree (ss);
_somFree (itr);
```

Original Class

somf_TIterator (overridden here)

Related Information

Methods: somfFirst, somfStartHere

somfPrevious Method

Purpose

Gets the previous object in a sorted sequence.

IDL Syntax

somf_MCollectible somfPrevious ( );

Description

The **somfPrevious** method determines the previous object in the **somf_TSortedSequence** collection that corresponds to the **somf_TSortedSequenceIterator** iterator used as the receiving object, and returns a pointer to it.

If the **somf_TSortedSequence** collection has changed (other than through the use of the **somfRemove** method of this iterator) since the last time the **somfLast** method was called, the iterator becomes invalid and will *fail* if asked to find the previous object. For example, if the collection's **somfAdd** method were called after starting to iterate through the collection, the iterator then would not allow iteration to continue. The iterator must be reset, and the easiest way to do that is to call the iterator's **somfLast** method and start over.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Thus, if any child of **somf_TSequenceIterator** is used with **somf_TPrimitiveLinkedListIterator**, then the name of the method will have to be fully qualified (for example: **somf_TSortedSequenceIterator_somfPrevious**). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you could have referenced this method as:

```
itr->somfPrevious(ev);
```

Parameters

receiver A pointer to an object of class **somf_TSortedSequenceIterator.**

ev A pointer to the **Environment** structure for the calling method.

Return Value

There are two possible valid return values for this method:

somf_MCollectible
A pointer to the previous object in the sorted sequence collection.

SOMF_NIL The beginning of the collection has been reached.

Example

```
somf_TSortedSequence ss;
Environment *ev;
somf_TSortedSequenceIterator itr;
<Your class which inherits from somf_MOrderableCollectible> itrobj;

ev = somGetGlobalEnvironment();

ss = somf_TSortedSequenceNew();
itr = somf_TSortedSequenceIteratorNew();
_somfTSortedSequenceIteratorInit(itr, ev, ss);
```

```
/* Go to the next to the last object in ss */
somf_TSortedSequenceIterator_somfLast(itr,ev);
itrobj =
      (<Your class which inherits from somf_MOrderableCollectible>*)
       _somfPrevious(itr,ev);

_somFree (ss);
_somFree (itr);
```

Original Class

somf_TSequenceIterator (overridden here)

Related Information

Methods: somfLast

somfRemove Method

Purpose

Removes the current object (the one just returned by a **somfFirst** or **somfNext** method) from a sorted sequence.

IDL Syntax

void somfRemove ( );

Description

The **somfRemove** method removes the current object (the one just returned by **somfFirst**, **somfNext, somfLast,** or **somfPrevious**) from the sorted sequence represented by the receiving object.

The **somfRemove** method is the only way to remove an object from a sorted sequence during iteration. However, if multiple iterators are in process, all other iterators are invalidated, just as if some other kind of change had occurred in the sorted sequence.

If the collection has changed (other than through the use of the **somfRemove** method of this iterator) since the last time **somfFirst** or **somfLast** was called, this method will *fail*.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. **somfRemove** is a method name declared in multiple parents (for example: **somf_TCollection**, **somf_THashTable**, **somf_TIterator**, etc.) You will probably have to fully qualify the method name (as **somf_TSortedSequenceIterator_somfRemove,** for example). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you could have referenced this method as:

```
itr->somfRemove(ev);
```

Parameters

receiver	A pointer to an object of class **somf_TSortedSequenceIterator.**
ev	A pointer to the **Environment** structure for the calling method.

Return Value

None.

Example

```
somf_TSortedSequence ss;
Environment *ev;
somf_TSortedSequenceIterator itr;
<Your class which inherits from somf_MOrderableCollectible> itrobj;

ev = somGetGlobalEnvironment();

ss = somf_TSortedSequenceNew();
itr = somf_TSortedSequenceIteratorNew();
_somfTSortedSequenceIteratorInit(itr, ev, ss);


/* Use the Iterator's Remove to remove the next to the last object
*/
itrobj =
      (<Your class which inherits from somf_MOrderableCollectible>*)
      somf_TSortedSequenceIterator_somfLast(itr,ev);
```

```
itrobj =
      (<Your class which inherits from somf_MOrderableCollectible>*)
       _somfPrevious(itr,ev);
somf_TSortedSequenceIterator_somfRemove(itr,ev);

_somFree (ss);
_somFree (itr);
```

Original Class

somf_TIterator (overridden here)

somfStartHere Method

Purpose

Begins Iterating through a **somf_TSortedSequence**, starting at a given object `obj`, rather than at the front of the collection.

IDL Syntax

somf_MOrderableCollectible somfStartHere (in somf_MOrderableCollectible *obj*);

Description

The **somfStartHere** method begins Iterating through a **somf_TSortedSequence** collection that corresponds to the **somf_TSortedSequenceIterator** iterator used as the receiving object. Iteration begins at the given object `obj`, rather than at the front of the collection.

Parameters

receiver	A pointer to an object of class **somf_TSortedSequenceIterator.**
ev	A pointer to the **Environment** structure for the calling method.
obj	A pointer to the **somf_MOrderableCollectible** object where iteration will begin.

Return Value

This method returns a pointer to the **somf_MCollectible** object where iteration started. Or, SOMF_NIL is returned if the collection is empty.

Example

```
somf_TSortedSequence ss;
Environment *ev;
somf_TSortedSequenceIterator itr;
<Your Class that inherits from somf_MOrderableCollectible> obj;
<Your class which inherits from somf_MOrderableCollectible> itrobj;

ev = somGetGlobalEnvironment();

ss = somf_TSortedSequenceNew();
obj = <Your Class that inherits from somf_MOrderableCollectible>New();
itr = somf_TSortedSequenceIteratorNew();
_somfTSortedSequenceIteratorInit(itr, ev, ss);


/* Iterate through the TSortedSequence starting at obj */
itrobj =
      (<Your class which inherits from somf_MOrderableCollectible>*)
      _somfStartHere(itr,ev,obj);
while (itrobj != SOMF_NIL)
{
    /* Do something with itrobj */

    itrobj =
      (<Your class which inherits from somf_MOrderableCollectible>*)
       _somfNext(itr,ev);
}

_somFree (ss);
_somFree (itr);
```

Original Class

somf_TSortedSequenceIterator

Related Information

Methods: somfFirst, somfNext

somfTSortedSequenceIteratorInit Method

Purpose

Initializes a new **somf_TSortedSequenceIterator** object, given the **somf_TSortedSequence** collection over which it will iterate.

IDL Syntax

somf_TSortedSequenceIterator somfTSortedSequenceIteratorInit (
in somf_TSortedSequence *h***);**

Description

The **somfTSortedSequenceIteratorInit** method initializes a **somf_TSortedSequenceIterator** iterator, given the **somf_TSortedSequence** object over which iteration is needed.

Note: This is one of three ways to initialize a **somf_TSortedSequenceIterator** to point to an instance of a **somf_TSortedSequence**. One other way is to use the **somf_TSortedSequence** class's **somfCreateSequenceIterator** method described on page 308. The final way is to use **somf_TSortedSequence**'s **somfCreateIterator** method described on page 307.

Note: You cannot override this method.

Parameters

receiver	A pointer to an object of class **somf_TSortedSequenceIterator.**
ev	A pointer to the **Environment** structure for the calling method.
h	A pointer to the sorted sequence that the receiving object will iterate over.

Return Value

This method returns a pointer to an initialized **somf_TSortedSequenceIterator** object.

Example

```
somf_TSortedSequence ss;
Environment *ev;
somf_TSortedSequenceIterator itr;

ev = somGetGlobalEnvironment();

ss = somf_TSortedSequenceNew();
itr = somf_TSortedSequenceIteratorNew();
_somfTSortedSequenceIteratorInit(itr, ev, ss);

_somFree (ss);
_somFree (itr);
```

Original Class

somf_TSortedSequenceIterator

somf_TSortedSequenceNode Class

Description

The **somf_TSortedSequenceNode** class defines a node in a tree. Objects inserted into a node must be of the **somf_MOrderableCollectible** class. Each node contains a key (the **somf_MOrderableCollectible** object) and links to a left child and a right child. An object of class **somf_TSortedSequenceNode** is used (transparently) by the **somf_TSortedSequence** class for each node of a sorted sequence collection. The **somf_TSortedSequenceNode** object provides the "linkability" (that is, the left and right links) to its two adjacent nodes in the collection.

The **somf_TSortedSequenceNode** class and methods will probably be of interest to programmers only in two situations: (a) if you are creating a new class that needs linkable nodes between objects of the class, or (b) if you are creating a new class that inherits from **somf_TSortedSequence,** and it would be appropriate to override some method(s) of the **somf_TSortedSequence** class to define additional functionality for those methods.

When you link, include the following library reference to get access to this class: **somtk**

This class is not thread-safe. Even if you put semaphores around your calls to this class's methods, different tasks should not be setting the values of the node. That situation is too prone to having multiple tasks setting conflicting values, leaving the instance in an unacceptable state.

This class is reentrant.

File Stem

tssnode

Base

SOMObject

Metaclass

SOMClass

Ancestor Classes

SOMObject

New Methods

somfGetLeftChild
somfGetRightChild
somfGetParent
somfGetKey
somfGetRed
somfSetParent
somfSetLeftChild
somfSetRightChild
somfSetKey
somfSetRed
somfSetRedOn
somfTSortedSequenceNodeInitTMT
somfTSortedSequenceNodeInitTM
somfTSortedSequenceNodeInitT

Overriding Methods

somInit

somfGetKey Method

Purpose

Gets the key to a node.

IDL Syntax

somf_MOrderableCollectible somfGetKey ( );

Description

The **somfGetKey** method determines the key to the node represented by the receiving object, and returns a pointer to the key.

Note: You cannot override this method.

Parameters

receiver A pointer to an object of class **somf_TSortedSequenceNode.**

ev A pointer to the **Environment** structure for the calling method.

Return Value

This method returns a pointer to the **somf_MOrderableCollectible** key.

Original Class

somf_TSortedSequenceNode

Related Information

Methods: somfSetKey

somfGetLeftChild Method

Purpose

Gets the left child of a node.

IDL Syntax

somf_TSortedSequenceNode somfGetLeftChild ( );

Description

The **somfGetLeftChild** method determines the left child of the node represented by the receiving object, and returns a pointer to the node.

Note: You cannot override this method.

Parameters

receiver A pointer to an object of class **somf_TSortedSequenceNode.**

ev A pointer to the **Environment** structure for the calling method.

Return Value

This method returns a pointer to the left child of the node.

Original Class

somf_TSortedSequenceNode

Related Information

Methods: somfSetLeftChild

somfGetParent Method

Purpose

Gets the parent of a node.

IDL Syntax

somf_TSortedSequenceNode somfGetParent ( );

Description

The **somfGetParent** method determines the parent of the node represented by the receiving object, and returns a pointer to the parent.

Note: You cannot override this method.

Parameters

receiver A pointer to an object of class **somf_TSortedSequenceNode.**

ev A pointer to the **Environment** structure for the calling method.

Return Value

This method returns a pointer to the parent of the node.

Original Class

somf_TSortedSequenceNode

Related Information

Methods: somfSetParent

somfGetRed Method

Purpose

Determines whether a node is red or black.

IDL Syntax

boolean somfGetRed ( );

Description

The **somfGetRed** method determines whether the node represented by the receiving object is red or black. Note: For a discussion of Red–Black Trees, you can refer to *Algorithms in C++* by Robert Sedgewick (Addison–Wesley Publishing Company, 1992).

Note: You cannot override this method.

Parameters

receiver	A pointer to an object of class **somf_TSortedSequenceNode.**
ev	A pointer to the **Environment** structure for the calling method.

Return Value

There are two possible valid return values for this method:

TRUE	The node is red.
FALSE	The node is black.

Original Class

somf_TSortedSequenceNode

Related Information

Methods: somfSetRed, somfSetRedOn

somfGetRightChild Method

Purpose

Gets the right child of a node.

IDL Syntax

somf_TSortedSequenceNode somfGetRightChild ( );

Description

The **somfGetRightChild** method determines the right child of the node represented by the receiving object, and returns a pointer to the node.

Note: You cannot override this method.

Parameters

receiver A pointer to an object of class **somf_TSortedSequenceNode.**

ev A pointer to the **Environment** structure for the calling method.

Return Value

This method returns the pointer to the right child of the node.

Original Class

somf_TSortedSequenceNode

Related Information

Methods: somfSetRightChild

somfSetKey Method

Purpose

Sets the key to a node, given a pointer to a key object.

IDL Syntax

void somfSetKey (in somf_MOrderableCollectible *k*);

Description

The **somfSetKey** method sets the key to the node represented by the receiving object, given a pointer to a **somf_MOrderableCollectible** object to be used as the key.

Note: You cannot override this method.

Parameters

receiver	A pointer to an object of class **somf_TSortedSequenceNode.**
ev	A pointer to the **Environment** structure for the calling method.
k	A pointer to the **somf_MOrderableCollectible** key.

Return Value

None.

Original Class

somf_TSortedSequenceNode

Related Information

Methods: somfGetKey

somfSetLeftChild Method

Purpose

Sets the left child of a node, given a pointer to an object that will be the left child.

IDL Syntax

void somfSetLeftChild (in somf_TSortedSequenceNode *n*);

Description

The **somfSetLeftChild** method sets the left child of the node represented by the receiving object, given a pointer to the **somf_TSortedSequenceNode** object to be used as the left child.

Note: You cannot override this method.

Parameters

receiver	A pointer to an object of class **somf_TSortedSequenceNode.**
ev	A pointer to the **Environment** structure for the calling method.
n	A pointer to the left child of the node.

Return Value

None.

Original Class

somf_TSortedSequenceNode

Related Information

Methods: somfGetLeftChild

somfSetParent Method

Purpose

Sets the parent of a node, given an object to be used as the parent node.

IDL Syntax

void somfSetParent (in somf_TSortedSequenceNode *n***);**

Description

The **somfSetParent** method sets the parent of the node represented by the receiving object, given a pointer to the **somf_TSortedSequenceNode** object to be used as the parent.

Note: You cannot override this method.

Parameters

receiver	A pointer to an object of class **somf_TSortedSequenceNode.**
ev	A pointer to the **Environment** structure for the calling method.
n	A pointer to the parent node of the receiving-object node.

Return Value

None.

Original Class

somf_TSortedSequenceNode

Related Information

Methods: somfGetParent

somfSetRed Method

Purpose

Sets a node to red or black.

IDL Syntax

void somfSetRed (in boolean *on***);**

Description

The **somfSetRed** method sets the node represented by the receiving object to either red or black, as determined by the boolean argument. Note: For a discussion of Red–Black Trees, you can refer to *Algorithms in C++* by Robert Sedgewick (Addison–Wesley Publishing Company, 1992).

Note: You cannot override this method.

Parameters

receiver	A pointer to an object of class **somf_TSortedSequenceNode**
ev	A pointer to the **Environment** structure for the calling method.
on	One of these two choices: TRUE The node is red; FALSE The node is black.

Return Value

None.

Original Class

somf_TSortedSequenceNode

Related Information

Methods: somfSetRedOn, somfGetRed

somfSetRedOn Method

Purpose

Sets a node to red.

IDL Syntax

void somfSetRedOn ( );

Description

The **somfSetRedOn** method sets the node represented by the receiving object to red, unconditionally. Note: For a discussion of Red–Black Trees, you can refer to *Algorithms in C++* by Robert Sedgewick (Addison–Wesley Publishing Company, 1992).

Note: You cannot override this method.

Parameters

receiver	A pointer to an object of class **somf_TSortedSequenceNode.**
ev	A pointer to the **Environment** structure for the calling method.

Return Value

None.

Original Class

somf_TSortedSequenceNode

Related Information

Methods: somfSetRed, somfGetRed

somfSetRightChild Method

Purpose

Sets the right child of a node, given a pointer to an object that will be the right child.

IDL Syntax

void somfSetRightChild (in somf_TSortedSequenceNode *n*);

Description

The **somfSetRightChild** method sets the right child of the node represented by the receiving object, given a pointer to the **somf_TSortedSequenceNode** object to be used as the right child.

Note: You cannot override this method.

Parameters

receiver	A pointer to an object of class **somf_TSortedSequenceNode.**
ev	A pointer to the **Environment** structure for the calling method.
n	A pointer to the right child of the node.

Return Value

None.

Original Class

somf_TSortedSequenceNode

Related Information

Methods: somfGetRightChild

somfTSortedSequenceNodeInitT Method

Purpose

Initializes a new **somf_TSortedSequenceNode** node, given its left child.

IDL Syntax

somf_TSortedSequenceNode somfTSortedSequenceNodeInitT (
in somf_TSortedSequenceNode *n1*);

Description

The **somfTSortedSequenceNodeInitT** method initializes the new node represented by the receiving object (a **somf_TSortedSequenceNode** object). The method call also specifies a **somf_TSortedSequenceNode** object to be used as the left child of the new node.

Note: You cannot override this method.

Parameters

receiver	A pointer to an object of class **somf_TSortedSequenceNode.**
ev	A pointer to the **Environment** structure for the calling method.
n1	A pointer to the left child of the new **somf_TSortedSequenceNode** object.

Return Value

This method returns a pointer to an initialized **somf_TSortedSequenceNode** object.

Original Class

somf_TSortedSequenceNode

Related Information

Methods: somfTSortedSequenceNodeInitTMT, somfTSortedSequenceNodeInitTM

somfTSortedSequenceNodeInitTM Method

Purpose

Initializes a new **somf_TSortedSequenceNode** node, given its left child and a key to the new node.

IDL Syntax

```
somf_TSortedSequenceNode  somfTSortedSequenceNodeInitTM (
                                    in somf_TSortedSequenceNode n1,
                                    in somf_MOrderableCollectible obj);
```

Description

The **somfTSortedSequenceNodeInitTM** method initializes the new node represented by the receiving object (a **somf_TSortedSequenceNode** object). The method call also specifies a **somf_TSortedSequenceNode** object to be used as the left child of the new node, and a **somf_MOrderableCollectible** object to be used as the key to the new node.

Note: You cannot override this method.

Parameters

receiver	A pointer to an object of class **somf_TSortedSequenceNode.**
ev	A pointer to the **Environment** structure for the calling method.
n1	A pointer to the left child of the new **somf_TSortedSequenceNode** object.
obj	A pointer to the key of the new **somf_TSortedSequenceNode** object.

Return Value

This method returns a pointer to an initialized **somf_TSortedSequenceNode** object.

Original Class

somf_TSortedSequenceNode

Related Information

Methods: somfTSortedSequenceNodeInitTMT, somfTSortedSequenceNodeInitT

somfTSortedSequenceNodeInitTMT Method

Purpose

Initializes a new **somf_TSortedSequenceNode** node, given a key to the new node and its left and right children.

IDL Syntax

```
somf_TSortedSequenceNode  somfTSortedSequenceNodeInitTMT (
                                    in somf_TSortedSequenceNode n1,
                                    in somf_MOrderableCollectible obj,
                                    in somf_TSortedSequenceNode n2);
```

Description

The **somfTSortedSequenceNodeInitTMT** method initializes the new node represented by the receiving object (a **somf_TSortedSequenceNode** object). The method call also specifies a **somf_MOrderableCollectible** object to be used as the key to the new node, as well as two **somf_TSortedSequenceNode** objects to be used as the left and right children of the new node.

Note: You cannot override this method.

Parameters

receiver	A pointer to an object of class **somf_TSortedSequenceNode.**
ev	A pointer to the **Environment** structure for the calling method.
n1	A pointer to the left child of the new **somf_TSortedSequenceNode** object.
obj	A pointer to the key of the new **somf_TSortedSequenceNode** object.
n2	A pointer to the right child of the new **somf_TSortedSequenceNode** object.

Return Value

This method returns a pointer to an initialized **somf_TSortedSequenceNode** object.

Original Class

somf_TSortedSequenceNode

Related Information

Methods: somfTSortedSequenceNodeInitTM, somfTSortedSequenceNodeInitT

Index

A

C

D

E

H

I

M

P

Q

S

U