somf_TDictionary Class

Description

This class represents a collection of (*key, value*) pairs (or associations). Because dictionaries are sometimes used to represent a bijective mapping, functions for retrieving a "key" given its corresponding "value" are provided, along with the usual access functions. (However, this process will probably be slow).

As for the **somf_THashTable** class, each entry in a **somf_TDictionary** is actually an object of the **somf_TAssoc** class that holds a (key, value) pair. In most cases, this use of a **somf_TAssoc** object is transparent to the user. However, you need to be aware of this **somf_TAssoc** usage, because some **somf_TDictionary** methods address the data as separate "key" and "value" parts of a pair, whereas other methods accept or return a single **somf_TAssoc** object representing the (key, value) pair.

The **somf_TDictionary** class is very similar to **somf_THashTable.** The primary difference is that **somf_TDictionary** inherits from **somf_TCollection**, whereas **somf_THashTable** does not. (Recall that all main collection classes except **somf_THashTable** inherit from the **somf_TCollection** class.) The other distinction is that the **somf_TDictionary** class uses the **somfIsEqual** method as its default comparison function, whereas **somf_THashTable** uses **somfIsSame.** Note that the **somf_TDictionary** class's use of **somfIsEqual** means that "equal" keys can only appear in the dictionary once.

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

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

Note: The **somf_TDictionary** 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 the initialization methods to change to the **somfIsSame** method.

Note: The **somf_TDictionary** class does *not* allow two entries have equal keys. Two separate, distinct entries can hash to the same hash value, but the original keys must not be equal.

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

tdict

Base

somf_TCollection

Metaclass

SOMClass

Ancestor Classes

somf_TCollection, somf_MCollectible, SOMObject

New Methods

somfDeleteAllKeys
somfDeleteAllValues
somfValueAt
somfKeyAtMF
somfKeyAtM
somfDeleteKey
somfAddKeyValuePairMMB,
somfAddKeyValuePairMM
somfSetHashFunction
somfGetHashFunction
somfCreateNewImplementationFLLL
somfCreateNewImplementationF
somfCreateNewImplementationFL
somfCreateNewImplementationFLL
somfCopyImplementation
somfAssign
somfTDictionaryInitFLL
somfTDictionaryInitFL
somfTDictionaryInitF
somfTDictionaryInitLLF
somfTDictionaryInitLL
somfTDictionaryInitL
somfTDictionaryInitD

Overriding Methods

somInit
somUninit
somfCreateIterator
somfRemove
somfRemoveAll
somfDeleteAll
somfCount
somfMember
somfAdd

somfAdd Method

Purpose

Adds a specified `obj` (representing a key, value pair) to the dictionary.

IDL Syntax

somf_MCollectible somfAdd (in somf_MCollectible *obj*);

Description

The **somfAdd** method adds a specified object `obj` to the dictionary represented by the receiving object. The added `obj` contains a (key, value) pair.

Do not be misled by this method's interface, which is inherited from the **somf_TCollection** class. The *only* objects you can add with **somfAdd** are (key, value) pairs of the **somf_TAssoc** class. You cannot use this interface to add a generic **somf_MCollectible** object.

Parameters

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

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

obj A pointer to an object that inherits from **somf_MCollectible** (specifically, a **somf_TAssoc** object) that will be added to the receiving object.

Return Value

somf_MCollectible

A pointer to the **somf_MCollectible** object that was added, provided a duplicate object does not exist. Otherwise, it returns a pointer to the value of the duplicate object, if *obj* could not be added because a duplicate is already in the collection. (Recall that an object of **somf_TDictionary** class will only accept one occurrence of an object where the **somfIsEqual** method would be TRUE.)

Example

```
somf_TDictionary d;
<Your Class which inherits from somf_MCollectible> obj;
<Your Class which inherits from somf_MCollectible> obj2;
somf_TAssoc tassoc;
Environment *ev;

ev = somGetGlobalEnvironment();

obj = <Your Class which inherits from somf_MCollectible>New();
obj2 = <Your Class which inherits from somf_MCollectible>New();
d = somf_TDictionaryNew();
tassoc = somf_TAssocNew();

_somfSetKey(tassoc, ev, obj);
_somfSetValue(tassoc, ev, obj2);
_somfAdd(d, ev, tassoc);

_somFree (d);
_somFree (obj);
_somFree (obj2);
_somFree (tassoc);
```

Original Class

somf_TCollection (overridden here)

Related Information

Methods: somfAddKeyValuePairMM, somfAddKeyValuePairMMB

somfAddKeyValuePairMM Method

Purpose

Adds a (key, value) pair to the receiving dictionary object, and returns a removed object (if removal was necessary).

IDL Syntax

somf_MCollectible somfAddKeyValuePairMM (
in somf_MCollectible *key***,**
in somf_MCollectible *val***);**

Description

The **somfAddKeyValuePairMM** method adds the specified (key, value) pair to the dictionary, even if there is an existing (key, value) pair that conflicts. The method also returns the value of the conflicting object (if any) that existed in the dictionary before this call.

Using the specified *key* and *val* arguments, this method transparently creates an object of the **somf_TAssoc** class, before adding the new (key, value) pair to the dictionary.

Parameters

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

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

key A pointer to a **somf_MCollectible** object that will be the key of the associated pair.

val A pointer to a **somf_MCollectible** object that will be the value of the associated pair.

Return Value

There are two possible valid return values for this method:

somf_MCollectible
A pointer to the **somf_MCollectible** value that had to be removed in order for a new (key, value) pair to be added.

SOMF_NIL No **somf_MCollectible** value had to be removed in order to add the pair.

Example

```
somf_TDictionary d;
<Your Class which inherits from somf_MCollectible> obj;
<Your Class which inherits from somf_MCollectible> obj2;
Environment *ev;

ev = somGetGlobalEnvironment();

obj = <Your Class which inherits from somf_MCollectible>New();
obj2 = <Your Class which inherits from somf_MCollectible>New();
d = somf_TDictionaryNew();

if (_somfAddKeyValuePairMM(d, ev, obj, obj2) != SOMF_NIL)
    somPrintf("\n problem adding obj,obj2 to d\n");

_somFree (d);
_somFree (obj);
_somFree (obj2);
```

Original Class

somf_TDictionary

Related Information

Methods: somfAddKeyValuePairMMB, somfAdd

somfAddKeyValuePairMMB Method

Purpose

Adds a (key, value) pair to a dictionary, unless the boolean argument prohibits replacement of a conflicting pair.

IDL Syntax

somf_MCollectible somfAddKeyValuePairMMB (
in somf_MCollectible *key***,**
in somf_MCollectible *val***,**
in boolean *replace***);**

Description

The **somfAddKeyValuePairMMB** method adds the stipulated (key, value) pair to the dictionary represented by the receiving object, unless the boolean argument *replace* prohibits this replacement.

If *replace*=TRUE, the (key, value) pair is added to the dictionary regardless of whether a conflicting pair exists. Otherwise, if *replace* = FALSE, then the (key, value) pair is added to the dictionary only if there is not a conflicting (key, value) pair.

Using the specified *key* and *val* arguments, this method transparently creates an object of the **somf_TAssoc** class, before adding the new (key, value) pair to the dictionary.

Parameters

receiver	A pointer to an object of class **somf_TDictionary.**
ev	A pointer to the **Environment** structure for the calling method.
key	A pointer to a **somf_MCollectible** object that is the key of the associated pair.
val	A pointer to a **somf_MCollectible** object that is the value of the associated pair.
replace	A boolean that indicates whether an existing pair with an identical key should be replaced.

Return Value

There are two possible valid return values for this method:

somf_MCollectible
A pointer to the **somf_MCollectible** value that had to be removed in order for a new (key, value) pair to be added.

SOMF_NIL — No **somf_MCollectible** value had to be removed in order to add the pair.

Example

```
somf_TDictionary d;
<Your Class which inherits from somf_MCollectible> obj2;
<Your Class which inherits from somf_MCollectible> obj3;
Environment *ev;

ev = somGetGlobalEnvironment();

obj2 = <Your Class which inherits from somf_MCollectible>New();
obj3 = <Your Class which inherits from somf_MCollectible>New();
d = somf_TDictionaryNew();
```

```
if (_somfAddKeyValuePairMMB(d, ev, obj2, obj3, TRUE) != SOMF_NIL)
    somPrintf("\n problem adding obj2,obj3 to d\n");

_somFree (d);
_somFree (obj2);
_somFree (obj3);
```

Original Class

somf_TDictionary

Related Information

Methods: somfAddKeyValuePairMM, somfAdd

somfAssign Method

Purpose

Assigns a dictionary as being “equal” to a given source dictionary.

IDL Syntax

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

Description

The **somfAssign** method assigns the dictionary receiving object to be “equal” to the source dictionary object. 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_TDictionary** is used with any other main collection class, then the name of the method must be fully qualified (for example: **somf_TDictionary_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, obj);
```

Parameters

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

Return Value

None.

Example

```
somf_TDictionary d;
somf_TDictionary d2;
Environment *ev;

ev = somGetGlobalEnvironment();

d = somf_TDictionaryNew();
d2 = somf_TDictionaryNew();

/* Add some objects to d */

/* Assign d2 = d */
somf_TDictionary_somfAssign(d2,ev,d);

_somFree (d);
_somFree (d2);
```

Original Class

somf_TDictionary

somfCopyImplementation Method

Purpose

Returns a hash table that is a copy of the hash table in a given dictionary.

IDL Syntax

somf_THashTable somfCopyImplementation ( );

Description

The **somfCopyImplementation** method returns a hash table that is a copy of the hash table in the dictionary represented by the receiving object. Normally, a client program will not need to invoke this method.

Parameters

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

Return Value

This method returns a pointer to the new **somf_THashTable** initialized to look like the hash table of the receiving object.

Original Class

somf_TDictionary

somfCount Method

Purpose

Gets the number of objects in a dictionary.

IDL Syntax

long somfCount ( );

Description

The **somfCount** method determines the number of objects in the dictionary represented by the receiving object.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Since **somf_TDictionary** uses **somf_THashTable**, the name of the method must 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_TDictionary.**
ev	A pointer to the **Environment** structure for the calling method.

Return Value

This method returns the number of objects in the receiving object.

Example

```
somf_TDictionary d;
<your Class which inherits from somf_MCollectible> obj;
Environment *ev;

ev = somGetGlobalEnvironment();

d = somf_TDictionaryNew();
obj = <your Class which inherits from somf_MCollectible>New();

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

_somFree (d);
_somFree (obj);
```

Original Class

somf_TCollection (overridden here)

somfCreateIterator Method

Purpose

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

IDL Syntax

somf_TIterator somfCreateIterator ( );

Description

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

Note: This is one of two ways to initialize a **somf_TDictionaryIterator** to point to an instance of a **somf_TDictionary**. The other way is to use the **somf_TDictionaryIterator**'s initializer method described on page 153.

Parameters

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

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

Return Value

This method returns a pointer to the new iterator.

Example

```
somf_TDictionary d;
Environment *ev;
somf_TDictionaryIterator itr;

ev = somGetGlobalEnvironment();

d = somf_TDictionaryNew();
itr = (somf_TDictionaryIterator*) _somfCreateIterator(d,ev);

_somFree (d);
_somFree (itr);
```

Original Class

somf_TCollection (overridden here)

somfCreateNewImplementationF Method

Purpose

Creates a new hash table for a dictionary, given its comparison test method.

IDL Syntax

somf_THashTable somfCreateNewImplementationF (
in somf_MCollectibleCompareFn *testfn***);**

Description

The **somfCreateNewImplementationF** method creates a new hash table for the dictionary represented by the receiving object. The method also includes an argument that defines the comparison test method applied to current/potential dictionary objects.

Normally, a client program does not invoke this method. However, if you create a new class that inherits from this class, you might consider overriding this method in order to customize how a **somf_TDictionary** object creates a new implementation.

When a **somfCreateNewImplementation...** method does not include arguments for the dictionary's table size, growth rate or rehash threshold, a default number of (key, value) pairs is assumed for the initial size, and the table subsequently grows by a default number of pairs once the dictionary contains a number of pairs that approaches the current table size.

Parameters

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

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_TDictionary** 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_TDictionary** object.

Return Value

This method returns a pointer to the new hash table.

Example

```
somf_THashTable ht2;
somf_TDictionary d;
Environment *ev;

ev = somGetGlobalEnvironment();
d = somf_TDictionaryNew();

ht2 = _somfCreateNewImplementationF
       (d, ev, somf_MCollectibleClassData.somfIsEqual);

_somFree (d);
_somFree (ht2);
```

Original Class

somf_TDictionary

Related Information

Methods: somfCreateNewImplementationFLLL, somfCreateNewImplementationFLL, somfCreateNewImplementationFL

somfCreateNewImplementationFL Method

Purpose

Creates a new hash table for a dictionary, given its comparison test method and its initial table size.

IDL Syntax

somf_THashTable somfCreateNewImplementationFL (
in somf_MCollectibleCompareFn *testfn*,
in long *tablesize*);

Description

The **somfCreateNewImplementationFL** method creates a new hash table for the dictionary represented by the receiving object. The method includes arguments that define the comparison test method applied to current/potential dictionary objects, and the initial size of the hash table.

Normally, a client program does not invoke this method. However, if you create a new class that inherits from this class, you might consider overriding this method in order to customize how a **somf_TDictionary** object creates a new implementation.

When a **somfCreateNewImplementation...** method does not include arguments for the dictionary's growth rate or rehash threshold, the initial table size will grow by a default number of pairs once the table contains a number of pairs that approaches the specified size.

Parameters

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

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_TDictionary** 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_TDictionary** object.

tablesize — The initial size of the hash table in the dictionary, expressed as the number of (key, value) pairs to expect.

Return Value

This method returns a pointer to the new hash table.

Example

```
somf_THashTable ht3;
somf_TDictionary d;
Environment *ev;

ev = somGetGlobalEnvironment();
d = somf_TDictionaryNew();

ht3 = _somfCreateNewImplementationFL
       (d, ev, somf_MCollectibleClassData.somfIsEqual, 23);

_somFree (d);
_somFree (ht3);
```

Original Class

somf_TDictionary

Related Information

Methods: somfCreateNewImplementationFLLL, somfCreateNewImplementationFLL, somfCreateNewImplementationF

somfCreateNewImplementationFLL Method

Purpose

Creates a new hash table for a dictionary, given its comparison test method, the initial table size, and the table's growth rate.

IDL Syntax

somf_THashTable somfCreateNewImplementationFLL (
in somf_MCollectibleCompareFn *testfn***,**
in long *tablesize***,**
in long *rate***);**

Description

The **somfCreateNewImplementationFLL** method creates a new hash table for the dictionary represented by the receiving object. The method includes arguments that define the comparison test method applied to current/potential dictionary objects, the initial table size, and the table's growth rate.

Normally, a client program does not invoke this method. However, if you create a new class that inherits from this class, you might consider overriding this method in order to customize how a **somf_TDictionary** object creates a new implementation.

When a **somfCreateNewImplementation...** method does not include an argument for the dictionary's rehash threshold, the initial table size will increment by the number of pairs given as the growth rate, once the table contains a number of pairs that approaches the specified size.

Parameters

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

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_TDictionary** 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_TDictionary** object.

tablesize The initial size of the hash table in the dictionary, expressed as the number of (key, value) pairs to expect.

rate The growth rate of the hash table in the dictionary, expressed as the number of (key, value) pairs by which to increment the allocated size when growth occurs.

Return Value

This method returns a pointer to the new hash table.

Example

```
somf_THashTable ht4;
somf_TDictionary d;
Environment *ev;
```

```
ev = somGetGlobalEnvironment();
d = somf_TDictionaryNew();

ht4 = _somfCreateNewImplementationFLL
       (d, ev, somf_MCollectibleClassData.somfIsEqual, 23, 20);

_somFree (d);
_somFree (ht4);
```

Original Class

somf_TDictionary

Related Information

Methods: somfCreateNewImplementationFLLL, somfCreateNewImplementationFL, somfCreateNewImplementationF

somfCreateNewImplementationFLLL Method

Purpose

Creates a new hash table for a dictionary, given its comparison test method, the initial table size, the table's growth rate, and the table's rehash threshold.

IDL Syntax

```
somf_THashTable  somfCreateNewImplementationFLLL (
                                    in somf_MCollectibleCompareFn testfn,
                                    in long tablesize,
                                    in long rate,
                                    in long threshold);
```

Description

The **somfCreateNewImplementationFLLL** method creates a new hash table for the dictionary represented by the receiving object. The hash table is fully specified by arguments that define the comparison test method applied to current/potential dictionary objects, the initial table size, the table's growth rate, and the table's rehash threshold.

Normally, a client program does not invoke this method. However, if you create a new class that inherits from this class, you might consider overriding this method in order to customize how a **somf_TDictionary** object creates a new implementation.

Parameters

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

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_TDictionary** 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_TDictionary** object.

tablesize The initial size of the hash table in the dictionary, expressed as the number of (key, value) pairs to expect.

rate The growth rate of the hash table in the dictionary, expressed as the number of (key, value) pairs by which to increment the allocated size when growth occurs.

threshold The rehash threshold of the hash table in the dictionary, expressed as the percentage of how full the dictionary may become before it grows in allocated size.

Return Value

This method returns a pointer to the new hash table.

Example

```
somf_THashTable ht1;
somf_TDictionary d;
Environment *ev;

ev = somGetGlobalEnvironment();
d = somf_TDictionaryNew();

ht1 = _somfCreateNewImplementationFLLL
       (d, ev, somf_MCollectibleClassData.somfIsEqual, 23, 20, 80);

_somFree (d);
_somFree (ht1);
```

Original Class

somf_TDictionary

Related Information

Methods: somfCreateNewImplementationFLL, somfCreateNewImplementationFL, somfCreateNewImplementationF

somfDeleteAll Method

Purpose

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

IDL Syntax

void somfDeleteAll ( );

Description

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

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. Since **somf_TDictionary** uses **somf_THashTable**, then the name of this method must 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_TDictionary.**
ev	A pointer to the **Environment** structure for the calling method.

Return Value

None.

Example

```
somf_TDictionary d;
Environment *ev;

ev = somGetGlobalEnvironment();

d = somf_TDictionaryNew();

/* Add some objects in the somf_TDictionary */

/* Remove all the objects AND DELETE THEM */
somf_TDictionary_somfDeleteAll(d,ev);

_somFree (d);
```

Original Class

somf_TCollection (overridden here)

Related Information

Methods: somfDeleteAllKeys, somfDeleteAllValues, somfDeleteKey

somfDeleteAllKeys Method

Purpose

Removes all of the (key, value) pairs from a dictionary. The procedure resets the count to zero and calls the destructor on every key in the dictionary.

IDL Syntax

void somfDeleteAllKeys ( );

Description

The **somfDeleteAllKeys** method removes all of the (key, value) pairs from a dictionary. The procedure resets the count to zero and calls the destructor on every key in the dictionary. However, the program still owns the objects representing the values of the (key, value) pairs.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Since **somf_TDictionary** uses **somf_THashTable**, then the name of this method must be fully qualified (as **somf_TDictionary_somfDeleteAllKeys,** 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->somfDeleteAllKeys(ev);
```

Parameters

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

Return Value

None.

Example

```
somf_TDictionary d;
Environment *ev;

ev = somGetGlobalEnvironment();

d = somf_TDictionaryNew();

/* Add some objects in the somf_TDictionary */

/* Remove all the objects AND DELETE ALL THE KEYS */
somf_TDictionary_somfDeleteAllKeys(d,ev);

_somFree (d);
```

Original Class

somf_TDictionary

Related Information

Methods: somfDeleteAll, somfDeleteAllValues, somfDeleteKey

somfDeleteAllValues Method

Purpose

Removes all of the (key, value) pairs from a dictionary. The procedure resets the count to zero and calls the destructor on every value in the hash table.

IDL Syntax

void somfDeleteAllValues ( );

Description

The **somfDeleteAllValues** method removes all of the (key, value) pairs from a dictionary. The procedure resets the count to zero and calls the destructor on every value in the hash table. However, the program still owns the objects representing the keys of the (key, value) pairs.

Because a dictionary only contains *pointers* to objects (rather than the objects themselves), **somfDeleteAllValues** can cause a problem if a pointer to an object appears more than once. For example, if pointer 'A' exists 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 more than once, you should consider using **somfRemoveAll** to remove the objects from the dictionary 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. Since **somf_TDictionary** uses **somf_THashTable**, the name of this method will have to be fully qualified (as **somf_TDictionary_somfDeleteAllValues,** 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->somfDeleteAllValues(ev);
```

Parameters

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

Return Value

None.

Example

```
somf_TDictionary d;
Environment *ev;

ev = somGetGlobalEnvironment();

d = somf_TDictionaryNew();

/* Add some objects in the somf_TDictionary */

/* Remove all the objects AND DELETE ALL THE VALUES */
somf_TDictionary_somfDeleteAllValues(d,ev);

_somFree (d);
```

Original Class

somf_TDictionary

Related Information

Methods: somfDeleteAllKeys, somfDeleteAll, somfDeleteKey

somfDeleteKey Method

Purpose

Deletes a specified key from the associated (key, value) pair, and removes the (key, value) pair from the dictionary.

IDL Syntax

somf_MCollectible somfDeleteKey (in somf_MCollectible *key***);**

Description

The **somfDeleteKey** method deletes the specified key from the associated (key, value) pair. The method also removes the (key, value) pair from the dictionary represented by the receiving object. The method returns a pointer to the value from the (key, value) pair.

Parameters

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

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

key A pointer to a **somf_MCollectible** object that is the key of the associated pair to be deleted.

Return Value

There are two possible valid return values for this method:

somf_MCollectible
A pointer to the value that was removed because the key was deleted.

SOMF_NIL The key object was not found.

Example

```
somf_TDictionary d;
<Your Class which inherits from somf_MCollectible> obj;
Environment *ev;

ev = somGetGlobalEnvironment();

obj = <Your Class which inherits from somf_MCollectible>New();
d = somf_TDictionaryNew();

/* Remove the key,value pair AND DELETE THE KEY */
if (_somfDeleteKey(d, ev, obj) == SOMF_NIL)
    somPrintf(" Why did DeleteKey say obj wasn't in d? \n");

_somFree (d);
```

Original Class

somf_TDictionary

Related Information

Methods: somfDeleteAllKeys, somfDeleteAllValues, somfDeleteAll

somfGetHashFunction Method

Purpose

Returns a pointer to the hash function used by a given dictionary.

IDL Syntax

somf_MCollectibleHashFn somfGetHashFunction ( );

Description

The **somfGetHashFunction** method returns a pointer to the hash function used by the dictionary that is the method's receiving object.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Since **somf_TDictionary** uses **somf_THashTable**, the name of this method will have to be fully qualified (as **somf_TDictionary_somfGetHashFunction,** 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->somfGetHashFunction(ev);
```

Parameters

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

Return Value

This method returns a pointer to the **somfHash** method used by this dictionary.

Example

```
somf_TDictionary d;
Environment *ev;

ev = somGetGlobalEnvironment();

d = somf_TDictionaryNew();

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

_somFree (d);
```

Original Class

somf_TDictionary

Related Information

Methods: somfSetHashFunction

somfKeyAtM Method

Purpose

Gets a dictionary's first key that has a specified *val* as its associated value. Note: This method involves a slow search.

IDL Syntax

somf_MCollectible somfKeyAtM (in somf_MCollectible *val*);

Description

The **somfKeyAtM** method finds the key of the first (key, value) pair whose value is the specified argument ***val,*** and returns a pointer to the key. Note that this method involves a slow search of the dictionary.

Parameters

receiver	A pointer to an object of class **somf_TDictionary.**
ev	A pointer to the **Environment** structure for the calling method.
val	A pointer to a **somf_MCollectible** that is the value to be searched for.

Return Value

There are two possible valid return values for this method:

somf_MCollectible
: A pointer to the dictionary's first key that has *val* as the value of the associated (key, value) pair.

SOMF_NIL
: The value was not found in the dictionary.

Example

```
somf_TDictionary d;
<your Class which inherits from somf_MCollectible> key;
<your Class which inherits from somf_MCollectible> value;
Environment *ev;

ev = somGetGlobalEnvironment();

d = somf_TDictionaryNew();
value = <your Class which inherits from somf_MCollectible>New();

/* Add a lot of objects to d */

key = _somfKeyAtM(d, ev, value);
if (key == SOMF_NIL)
   somPrintf(" value is no longer in d\n");
else
{
    /* do something with the key */
}

_somFree (d);
_somFree (value);
```

Original Class

somf_TDictionary

Related Information

Methods: somfKeyAtMF

somfKeyAtMF Method

Purpose

Gets a dictionary's first key that has a specified *val* as its associated value. The method includes an argument specifying the method to be used for comparing the values. Note: This method involves a slow search.

IDL Syntax

somf_MCollectible somfKeyAtMF (
in somf_MCollectible *val***,**
in somf_MCollectibleCompareFn *testfn***);**

Description

The **somfKeyAtMF** method finds the key of the first (key, value) pair whose value is the specified argument *val***,** and returns a pointer to the key. The method includes an argument that specifies whether **somfIsEqual** or **somfIsSame** should be used to compare the values. Note that this method involves a slow search of the dictionary.

Parameters

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

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

val A pointer to a **somf_MCollectible** object that is the value to be searched for.

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_TDictionary** 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_TDictionary** object.

Return Value

There are two possible valid return values for this method:

somf_MCollectible
A pointer to the dictionary's first key that has *val* as the value of the associated (key, value) pair.

SOMF_NIL The value was not found in the dictionary.

Example

```
somf_TDictionary d;
<your Class which inherits from somf_MCollectible> key;
<your Class which inherits from somf_MCollectible> value;
Environment *ev;

ev = somGetGlobalEnvironment();

d = somf_TDictionaryNew();
value = <your Class which inherits from somf_MCollectible>New();
```

```
/* Add a lot of objects to d */

key = _somfKeyAtMF(d, ev, value,
                           somf_MCollectibleClassData.somfIsEqual);

if (key == SOMF_NIL)
    somPrintf(" value is no longer in d\n");
else
{
    /* do something with the key */
}

_somFree (d);
_somFree (value);
```

Original Class

somf_TDictionary

Related Information

Methods: somfKeyAtM

somfMember Method

Purpose

Gets the key of a (key, value) pair in the dictionary, if it is found.

IDL Syntax

somf_MCollectible somfMember (in somf_MCollectible *key***);**

Description

The **somfMember** method determines whether the (key, value) pair corresponding to a specified `key` is in the dictionary and, if so, returns a pointer to the key object.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Since **somf_TDictionary** uses **somf_THashTable**, then the name of the method will have to be fully qualified (as **somf_TDictionary_somfMember,** 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->somfMember(ev, obj);
```

Parameters

receiver	A pointer to an object of class **somf_TDictionary.**
ev	A pointer to the **Environment** structure for the calling method.
key	A pointer to the **somf_MCollectible** key of the (key, value) pair that may or may not be in the dictionary.

Return Value

There are two possible valid return values for this method:

somf_MCollectible
: A pointer to the key of the (key, value) pair that the method determined as the member.

SOMF_NIL
: The object was not found.

Example

```
somf_TDictionary d;
<your Class which inherits from somf_MCollectible> obj;
Environment *ev;

ev = somGetGlobalEnvironment();

d = somf_TDictionaryNew();
obj = <your Class which inherits from somf_MCollectible>New();

/* Add some objects to d */

/* See if obj is in d */
if (somf_TDictionary_somfMember(d, ev, obj) == SOMF_NIL)
    somPrintf("\n obj is NOT in d\n");
else
    somPrintf("\n obj IS in d\n");

_somFree (d);
```

Original Class

somf_TCollection (overridden here)

somfRemove Method

Purpose

Removes from the dictionary the (key, value) pair associated with a given key.

IDL Syntax

somf_MCollectible somfRemove (in somf_MCollectible *key***);**

Description

The **somfRemove** method removes from the dictionary the (key, value) pair associated with the specified *key* object.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Since **somf_TDictionary** uses **somf_THashTable**, then the name of the method will have to be fully qualified (as **somf_TDictionary_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_TDictionary.**
ev	A pointer to the **Environment** structure for the calling method.
key	A pointer to the **somf_MCollectible** object representing the key to be removed from the dictionary.

Return Value

There are two possible valid return values for this method:

somf_MCollectible
: A pointer to the value of the (key, value) pair that was removed.

SOMF_NIL
: The *key* object was not found.

Example

```
somf_TDictionary d;
<your Class which inherits from somf_MCollectible> key;
Environment *ev;

ev = somGetGlobalEnvironment();

d = somf_TDictionaryNew();
key = <your Class which inherits from somf_MCollectible>New();

/* Add a lot of objects to d */

if (somf_TDictionary_somfRemove(d, ev, key) == SOMF_NIL)
    somPrintf(" Why did Remove say key was not removed?\n");

_somFree (d);
_somFree (key);
```

Original Class

somf_TCollection (overridden here)

Related Information

Methods: somfRemoveAll

somfRemoveAll Method

Purpose

Removes all of the (key, value) pairs from a dictionary.

IDL Syntax

void somfRemoveAll ( );

Description

The **somfRemoveAll** method removes all of the (key, value) pairs from the dictionary that is the receiving object.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Since **somf_TDictionary** uses **somf_THashTable**, 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_TDictionary.**
ev	A pointer to the **Environment** structure for the calling method.

Return Value

None.

Example

```
somf_TDictionary d;
Environment *ev;

ev = somGetGlobalEnvironment();

d = somf_TDictionaryNew();

/* Add a lot of objects to d */

/* remove all the objects from d */
somf_TDictionary_somfRemoveAll(d,ev);

_somFree (d);
```

Original Class

somf_TCollection (overridden here)

Related Information

Methods: somfRemove

somfSetHashFunction Method

Purpose

Sets a dictionary's hash-function pointer to a given method.

IDL Syntax

void somfSetHashFunction (in somf_MCollectibleHashFn *fn*);

Description

The **somfSetHashFunction** method sets the pointer for the dictionary's hash function to the specified method *fn*. By default, this pointer is set to **somf_MCollectible**'s **somfHash** method (which is usually overridden in the objects that are added to the hash table). Normally, a client program does not invoke this method.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Since **somf_TDictionary** uses **somf_THashTable**, the name of this method will have to be fully qualified (as **somf_TDictionary_somfSetHashFunction,** 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->somfSetHashFunction(ev);
```

Parameters

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

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_TDictionary** 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_TDictionary** object.

Return Value

None.

Example

```
somf_TDictionary d;
Environment *ev;

ev = somGetGlobalEnvironment();

d = somf_TDictionaryNew();

somf_TDictionary_somfSetHashFunction(d, ev,
                                     somf_MCollectibleClassData.somfHash);

_somFree (d);
```

Original Class

somf_TDictionary

Related Information

Methods: somfGetHashFunction

somfTDictionaryInitD Method

Purpose

Initializes a new dictionary, setting it equal to another specified dictionary.

IDL Syntax

somf_TDictionary somfTDictionaryInitD (in somf_TDictionary *dictionary***);**

Description

The **somfTDictionaryInitD** method initializes the new dictionary represented by the receiving object. The method also sets the new dictionary equal to another specified dictionary. This implies that the instance data of the new dictionary will be set equal to those of the source dictionary.

Note: You cannot override this method.

Parameters

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

Return Value

This method returns a pointer to an initialized **somf_TDictionary** object representing the new dictionary.

Example

```
somf_TDictionary d2;
somf_TDictionary d7;
Environment *ev;

ev = somGetGlobalEnvironment();

d2 = somf_TDictionaryNew();
d7 = somf_TDictionaryNew();
_somfTDictionaryInitD(d7, ev, d2);

_somFree (d2);
_somFree (d7);
```

Original Class

somf_TDictionary

Related Information

Methods: somfTDictionaryInitFLL, somfTDictionaryInitFL, somfTDictionaryInitF, somfTDictionaryInitLLF, somfTDictionaryInitLL, somfTDictionaryInitL

somfTDictionaryInitF Method

Purpose

Initializes a new dictionary, given its comparison test method.

IDL Syntax

somf_TDictionary somfTDictionaryInitF (in somf_MCollectibleCompareFn *testfn***);**

Description

The **somfTDictionaryInitF** method initializes a new dictionary, given its comparison test method.

When a **somfDictionaryInit...** method does not include arguments for the dictionary's table size or growth rate, a default number of (key, value) pairs is assumed for the initial size, and the table subsequently grows by a default number of pairs once the dictionary contains a number of pairs approaching the current table size.

Note: You cannot override this method.

Parameters

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

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_TDictionary** 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_TDictionary** object.

Return Value

This method returns a pointer to an initialized **somf_TDictionary** object representing the new dictionary.

Example

```
somf_TDictionary d3;
Environment *ev;

ev = somGetGlobalEnvironment();

d3 = somf_TDictionaryNew();
_somfTDictionaryInitF(d3, ev,
                      somf_MCollectibleClassData.somfIsEqual);

_somFree (d3);
```

Original Class

somf_TDictionary

Related Information

Methods: somfTDictionaryInitFLL, somfTDictionaryInitFL, somfTDictionaryInitLLF, somfTDictionaryInitLL, somfTDictionaryInitL, somfTDictionaryInitD

somfTDictionaryInitFL Method

Purpose

Initializes a new dictionary, given its comparison test method and its initial size.

IDL Syntax

somf_TDictionary somfTDictionaryInitFL (
in somf_MCollectibleCompareFn *testfn***,**
in long *sizeHint***);**

Description

The **somfTDictionaryInitFL** method initializes a new dictionary, given its comparison test method and its initial size.

When a **somfDictionaryInit...** method does not include an argument for the dictionary's growth rate, the initial table size will grow by a default number of pairs once the table contains a number of pairs that approaches the specified size.

Note: You cannot override this method.

Parameters

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

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_TDictionary** 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_TDictionary** object.

sizeHint The initial size of the dictionary, expressed as the number of (key, value) pairs to expect.

Return Value

This method returns a pointer to an initialized **somf_TDictionary** object representing the new dictionary.

Example

```
somf_TDictionary d2;
Environment *ev;

ev = somGetGlobalEnvironment();

d2 = somf_TDictionaryNew();
_somfTDictionaryInitFL(d2, ev,
                       somf_MCollectibleClassData.somfIsEqual, 8);

_somFree (d2);
```

Original Class

somf_TDictionary

Related Information

Methods: somfTDictionaryInitFLL, somfTDictionaryInitF, somfTDictionaryInitLLF, somfTDictionaryInitLL, somfTDictionaryInitL, somfTDictionaryInitD

somfTDictionaryInitFLL Method

Purpose

Initializes a new dictionary, given its comparison test method, its initial size, and its initial growth rate. Note: This method is equivalent to the **somfTDictionaryInitLLF** method.

IDL Syntax

somf_TDictionary somfTDictionaryInitFLL (
in somf_MCollectibleCompareFn *testfn***,**
in long *sizeHint***,**
in long *growthRate***);**

Description

The **somfTDictionaryInitFLL** method initializes a new dictionary, given its comparison test method, its initial size, and its growth rate.

Note: You cannot override this method.

Parameters

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

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_TDictionary** 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_TDictionary** object.

sizeHint The initial size of the dictionary, expressed as the number of (key, value) pairs to expect.

growthRate The initial growth rate, expressed as the number of (key, value) pairs by which to increment the allocated size when growth occurs.

Return Value

This method returns a pointer to the initialized **somf_TDictionary** object representing the new dictionary.

Example

```
somf_TDictionary d1;
Environment *ev;

ev = somGetGlobalEnvironment();

d1 = somf_TDictionaryNew();
_somfTDictionaryInitFLL(d1, ev,
                        somf_MCollectibleClassData.somfIsEqual, 8, 8);

_somFree (d1);
```

Original Class

somf_TDictionary

Related Information

Methods: somfTDictionaryInitFL, somfTDictionaryInitF, somfTDictionaryInitLLF, somfTDictionaryInitLL, somfTDictionaryInitL, somfTDictionaryInitD

somfTDictionaryInitL Method

Purpose

Initializes a new dictionary, given its initial size.

IDL Syntax

somf_TDictionary somfTDictionaryInitL (in long *sizeHint***);**

Description

The **somfTDictionaryInitL** method initializes a new dictionary, given its initial size.

When a **somfDictionaryInit...** method does not include an argument for the dictionary's growth rate, the initial table size will grow by a default number of pairs once the table contains a number of pairs that approaches the specified size. When a comparison method is not specified, the default **somfIsEqual** method is used unless the **somfSetHashFunction** method has changed it to **somfIsSame.**

Note: You cannot override this method.

Parameters

receiver	A pointer to an object of class **somf_TDictionary.**
ev	A pointer to the **Environment** structure for the calling method.
sizeHint	The initial size of the dictionary, expressed as the number of (key, value) pairs to expect.

Return Value

This method returns a pointer to an initialized **somf_TDictionary** object representing the new dictionary.

Example

```
somf_TDictionary d6;
Environment *ev;

ev = somGetGlobalEnvironment();

d6 = somf_TDictionaryNew();
_somfTDictionaryInitL(d6, ev, 8);

_somFree (d6);
```

Original Class

somf_TDictionary

Related Information

Methods: somfTDictionaryInitFLL, somfTDictionaryInitFL, somfTDictionaryInitF, somfTDictionaryInitLLF, somfTDictionaryInitLL, somfTDictionaryInitD

somfTDictionaryInitLL Method

Purpose

Initializes a new dictionary, given its initial size and its initial growth rate.

IDL Syntax

somf_TDictionary somfTDictionaryInitLL (
in long *sizeHint***,**
in long *growthRate***);**

Description

The **somfTDictionaryInitLL** method initializes a new dictionary, given its initial size and its initial growth rate. The default **somfIsEqual** method is used as the comparison method, unless the **somfSetHashFunction** method has changed it to **somfIsSame**.

Note: You cannot override this method.

Parameters

receiver	A pointer to an object of class **somf_TDictionary.**
ev	A pointer to the **Environment** structure for the calling method.
sizeHint	The initial size of the dictionary, expressed as the number of (key, value) pairs to expect.
growthRate	The initial growth rate, expressed as the number of (key, value) pairs by which to increment the allocated size when growth occurs.

Return Value

This method returns a pointer to an initialized **somf_TDictionary** object representing the new dictionary.

Example

```
somf_TDictionary d5;
Environment *ev;

ev = somGetGlobalEnvironment();

d5 = somf_TDictionaryNew();
_somfTDictionaryInitLL(d5, ev, 8, 8);

_somFree (d5);
```

Original Class

somf_TDictionary

Related Information

Methods: somfTDictionaryInitFLL, somfTDictionaryInitFL, somfTDictionaryInitF, somfTDictionaryInitLLF, somfTDictionaryInitL, somfTDictionaryInitD

somfTDictionaryInitLLF Method

Purpose

Initializes a new dictionary, given its initial size, its initial growth rate, and its comparison test method. Note: This method is equivalent to the **somfTDictionaryInitFLL** method.

IDL Syntax

somf_TDictionary somfTDictionaryInitLLF (
in long *sizeHint***,**
in long *growthRate***,**
in somf_MCollectibleCompareFn *testfn***);**

Description

The **somfTDictionaryInitLLF** method initializes a new dictionary, given its initial size, its initial growth rate, and its comparison test method.

Note: You cannot override this method.

Parameters

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

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

sizeHint The initial size of the dictionary, expressed as the number of (key, value) pairs to expect.

growthRate The initial growth rate, expressed as the number of (key, value) pairs by which to increment the allocated size when growth occurs.

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_TDictionary** 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_TDictionary** object.

Return Value

This method returns a pointer to an initialized **somf_TDictionary** object representing the new dictionary.

Example

```
somf_TDictionary d4;
Environment *ev;

ev = somGetGlobalEnvironment();

d4 = somf_TDictionaryNew();
_somfTDictionaryInitLLF(d4, ev, 8, 8,
                        somf_MCollectibleClassData.somfIsEqual);

_somFree (d4);
```

Original Class

somf_TDictionary

Related Information

Methods: somfTDictionaryInitFLL, somfTDictionaryInitFL, somfTDictionaryInitF, somfTDictionaryInitLL, somfTDictionaryInitL, somfTDictionaryInitD

somfValueAt Method

Purpose

Gets the value associated with a given key for a (key, value) pair in a dictionary.

IDL Syntax

somf_MCollectible somfValueAt (in somf_MCollectible *key***);**

Description

The **somfValueAt** method finds the value associated with the specified key for a (key, value) pair in a dictionary, and returns a pointer to the value.

Parameters

receiver	A pointer to an object of class **somf_TDictionary.**
ev	A pointer to the **Environment** structure for the calling method.
key	A pointer to a **somf_MCollectible** object that is the key to be searched for.

Return Value

There are two possible valid return values for this method:

somf_MCollectible
: A pointer to the **somf_MCollectible** object that is the value associated with the key.

SOMF_NIL
: The key was not found in the dictionary.

Example

```
somf_TDictionary d;
somf_MCollectible value;
<your Class which inherits from somf_MCollectible> key;
Environment *ev;

ev = somGetGlobalEnvironment();

d = somf_TDictionaryNew();
key = <your Class which inherits from somf_MCollectible>New();

/* Add a lot of objects to d */

value = _somfValueAt(d, ev, key);

_somFree (d);
_somFree (key);
```

Original Class

somf_TDictionary

somf_TDictionaryIterator Class

Description

The **somf_TDictionaryIterator** class defines an iterator for the **somf_TDictionary** class that will iterate over all of the objects in a dictionary.

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. Recall that each entry in a **somf_TDictionary** is actually an object of the **somf_TAssoc** class that holds a (key, value) pair. Thus, the **somfFirst** and **somfNext** methods in the **somf_TDictionaryIterator** class actually return **somf_TAssoc** objects, not simply objects of the **somf_MCollectible** class. You must handle the return values as if they were **somf_TAssoc**'s.

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

tdictitr

Base

somf_TIterator

Metaclass

SOMClass

Ancestor Classes

somf_TIterator, SOMObject

New Methods

somfTDictionaryIteratorInit

Overriding Methods

somInit
somUninit
somfFirst
somfNext
somfRemove

somfFirst Method

Purpose

Resets the iterator and returns the first (key, value) pair from a dictionary.

IDL Syntax

somf_MCollectible somfFirst ( );

Description

The **somfFirst** method resets the iterator and returns the first (key, value) pair from the dictionary of the dictionary iterator represented by the receiving object.

This resets the iterator to the beginning of the dictionary. This is true not only for the first time you use the iterator; it is also true if other operations on the dictionary cause the iterator to be invalidated. In the second case, the method also revalidates the iterator.

Do not be misled by this method's interface, which is inherited from the **somf_TIterator** class. The *only* objects returned with **somfFirst** are (key, value) pairs of the **somf_TAssoc** class. You cannot use the return value as a generic **somf_MCollectible** 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_TDictionaryIterator_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_TDictionaryIterator.**
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 dictionary. Or, SOMF_NIL is returned if the collection is empty.

Example

```
somf_TDictionary d;
Environment *ev;
somf_TDictionaryIterator itr;
somf_TAssoc itrobj;
somf_MCollectible objk;
somf_MCollectible objv;

ev = somGetGlobalEnvironment();

d = somf_TDictionaryNew();
itr = somf_TDictionaryIteratorNew();
_somfTDictionaryIteratorInit(itr, ev, d);

/* Add some object to d */

/* Iterate through the TDictionary */
itrobj = somf_TDictionaryIterator_somfFirst(itr,ev);
```

```
while (itrobj != SOMF_NIL)
{
    objk = _somfGetKey(itrobj,ev);
    objv = _somfGetValue(itrobj,ev);

   /* Do something with objk or objv */

    itrobj = _somfNext(itr,ev);
}

_somFree (d);
_somFree (itr);
```

Original Class

somf_TIterator (overridden here)

Related Information

Methods: somfNext

somfNext Method

Purpose

Gets the next (key, value) pair in the dictionary of a given dictionary iterator.

IDL Syntax

somf_MCollectible somfNext ( );

Description

The **somfNext** method determines the next (key, value) pair in the dictionary of the specified dictionary iterator. The method also returns a pointer to the next (key, value) pair, if found. Objects are retrieved in an order that reflects the "ordered-ness" of the dictionary (or the lack of ordering on the dictionary objects).

Do not be misled by this method's interface, which is inherited from the **somf_TIterator** class. The *only* objects returned with **somfNext** are (key, value) pairs of the **somf_TAssoc** class. You cannot use the return value as a generic **somf_MCollectible** object.

If the dictionary 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. Thus, if any child of **somf_TIterator** is used with an object of the **somf_TPrimitiveLinkedListIterator** class, then the name of the method must 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);
```

Parameters

receiver	A pointer to an object of class **somf_TDictionaryIterator.**
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 dictionary.

SOMF_NIL
: The end of the dictionary has been reached.

Example

```
somf_TDictionary d;
Environment *ev;
somf_TDictionaryIterator itr;
somf_TAssoc itrobj;
somf_MCollectible objk;
somf_MCollectible objv;

ev = somGetGlobalEnvironment();

d = somf_TDictionaryNew();
itr = somf_TDictionaryIteratorNew();
_somfTDictionaryIteratorInit(itr, ev, d);
```

```
/* Add some object to d */

/* Iterate through the TDictionary */
itrobj = somf_TDictionaryIterator_somfFirst(itr,ev);
while (itrobj != SOMF_NIL)
{
    objk = _somfGetKey(itrobj,ev);
    objv = _somfGetValue(obj,ev);

   /* Do something with objk or objv */

    itrobj = _somfNext(itr,ev);
}

_somFree (d);
_somFree (itr);
```

Original Class

somf_TIterator (overridden here)

Related Information

Methods: somfFirst

somfRemove Method

Purpose

Removes the current (key, value) pair (the one just returned by **somfFirst** or **somfNext**) from the dictionary.

IDL Syntax

void somfRemove ( );

Description

The **somfRemove** method removes the current (key, value) pair (the object just returned by **somfFirst** or **somfNext**) from the dictionary that corresponds to the dictionary iterator represented by the receiving object.

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

If the dictionary 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 (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_TDictionaryIterator.**
ev	A pointer to the **Environment** structure for the calling method.

Return Value

None.

Example

```
somf_TDictionary d;
Environment *ev;
somf_TDictionaryIterator itr;
somf_MCollectible itrobj;

ev = somGetGlobalEnvironment();

d = somf_TDictionaryNew();
itr = somf_TDictionaryIteratorNew();
_somfTDictionaryIteratorInit(itr, ev, d);

/* Add some objects to d */

/* Use the Iterator's Remove to remove the first object */
itrobj = somf_TDictionaryIterator_somfFirst(itr,ev);
somf_TDictionaryIterator_somfRemove(itr,ev);
```

```
_somFree (d);
_somFree (itr);
```

Original Class

somf_TIterator (overridden here)

somfTDictionaryIteratorInit Method

Purpose

Initializes a **somf_TDictionaryIterator** iterator for a specified dictionary.

IDL Syntax

somf_TDictionaryIterator somfTDictionaryIteratorInit (in somf_TDictionary *h*);

Description

The **somfTDictionaryIteratorInit** method initializes an iterator of **somf_TDictionaryIterator** class, given the **somf_TDictionary** dictionary over which iteration is needed.

Note: This is one of two ways to initialize a **somf_TDictionaryIterator** to point to an instance of a **somf_TDictionary** dictionary. The other way is to use the **somf_TDictionary** class's **somfCreateIterator** method described on page 114.

Note: You cannot override this method.

Parameters

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

Return Value

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

Example

```
somf_TDictionary d;
Environment *ev;
somf_TDictionaryIterator itr;

ev = somGetGlobalEnvironment();

d = somf_TDictionaryNew();
itr = somf_TDictionaryIteratorNew();
_somfTDictionaryIteratorInit(itr, ev, d);

_somFree (d);
_somFree (itr);
```

Original Class

somf_TDictionaryIterator

somf_THashTable Class

Description

Every hash table contains a set of (*key, value*) pairs that associate a key with a value. Hash tables provide fast lookup of a value when given its associated key, even if there are a large number of entries in the table. Methods are provided for the usual operations (insert, delete, and so forth) as well as for controlling when rehashing will occur, and the growth of the table when a rehash occurs.

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

As for the **somf_TDictionary** class, each entry in a **somf_THashTable** is actually an object of the **somf_TAssoc** class that holds a (key, value) pair. In most cases, this use of a **somf_TAssoc** object is transparent to the user. However, you need to be aware of this **somf_TAssoc** usage, because some **somf_THashTable** methods address the data as separate "key" and "value" parts of a pair, whereas other methods accept or return a single **somf_TAssoc** object representing the (key, value) pair.

The **somf_THashTable** class is very similar to **somf_TDictionary.** The primary difference is that **somf_THashTable** inherits directly from the **somf_MCollectible** class**,** whereas **somf_TDictionary** is another level down, inheriting from **somf_TCollection**. (Recall that all main collection classes except **somf_THashTable** inherit from the **somf_TCollection** class.) The other distinction is that the **somf_THashTable** class uses the **somfIsSame** method as its default comparison function, whereas **somf_TDictionary** uses **somfIsEqual.**

Objects inserted into a **somf_THashTable** collection *must* inherit from **somf_MCollectible**. In addition, they should override the **somfHash** method, and the **somfIsEqual** method. These methods are used internally by objects of the **somf_THashTable** class.

Because **somf_THashTable** takes **somf_MCollectible** objects as members, any class that inherits from **somf_MCollectible** can be a member of the hash table. This means, for example, that you can have a hash table containing **somf_TDeque** objects, or objects of any main collection class.

Note: The **somf_THashTable** class uses the **somfIsSame** method as the default comparison function. That is, if `key1="Bart"` and `key2="Bart"`, `key1` and `key2` are *not* the same. Only key1 is the same as key1. If you don't want to use the **somfIsSame** method to equate entries, use one of the initialization methods to change to the **somfIsEqual** method. Just be aware that if the comparison methods are changed, the objects inserted into the **somf_THashTable** *must* have **somfIsEqual** and **somfHash** overridden.

Note: This Hash Table does *not* allow two (key, value) pairs to have the same key. Two separate, distinct pairs can hash to the same hash value, but the instantiation of each original key must be unique.

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

thash

Base

somf_MCollectible

Metaclass

SOMClass

Ancestor Classes

somf_MCollectible, SOMObject

New Methods

somfCount
somfRemove
somfDelete
somfMember
somfRemoveAll
somfDeleteAll
somfDeleteAllKeys
somfDeleteAllValues
somfAddMMB
somfAddMM
somfGrow
somfRetrieve
somfSetGrowthRate
somfSetRehashThreshold
somfGetGrowthRate
somfGetRehashThreshold
somfSetHashFunction
somfGetHashFunction
somfAssign,
somfTHashTableInitFLLL
somfTHashTableInitFLL
somfTHashTableInitFL
somfTHashTableInitH

Overriding Methods

somInit
somUninit

somfAddMM Method

Purpose

Adds a (key, value) pair to the hash table. This method will replace a copy (a pair having the same key) if one already exists.

IDL Syntax

somf_MCollectible somfAddMM (
in somf_MCollectible *key***,**
in somf_MCollectible *value***);**

Description

The **somfAddMM** method adds the specified (key, value) pair to the hash table. If the hash table contains an existing (key,value) pair for *key*, the method removes the existing value, adds the new *value*, and returns the existing value of the conflicting (key, value) pair.

Using the specified *key* and *value* arguments, this method transparently creates an object of the **somf_TAssoc** class, before adding the new (key, value) pair to the hash table.

Note: You cannot override this method.

Parameters

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

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

key A pointer to a **somf_MCollectible** object that will be the key of the associated pair.

value A pointer to a **somf_MCollectible** object that will be the value of the associated pair.

Return Value

There are two possible valid return values for this method:

somf_MCollectible
A pointer to the **somf_MCollectible** object that was removed when *value* was inserted.

SOMF_NIL No object had to be removed to add the new (key, value) pair.

Example

```
somf_THashTable ht;
<Your Class which inherits from somf_MCollectible> obj;
<Your Class which inherits from somf_MCollectible> obj2;
Environment *ev;

ev = somGetGlobalEnvironment();

obj = <Your Class which inherits from somf_MCollectible>New();
obj2 = <Your Class which inherits from somf_MCollectible>New();
ht = somf_THashTableNew();

if (_somfAddMM(ht, ev, obj, obj2) != SOMF_NIL)
    somPrintf("\n problem adding obj,obj2 to ht\n");

_somFree (ht);
```

```
    _somFree (obj);
    _somFree (obj2);
```

Original Class

somf_THashTable

Related Information

Methods: somfAddMMB

somfAddMMB Method

Purpose

Adds a (key, value) pair to the hash table, unless the boolean argument prohibits replacement of a copy (a pair with the same key).

IDL Syntax

```
somf_MCollectible  somfAddMMB (
                                in somf_MCollectible key,
                                in somf_MCollectible value,
                                in boolean replace);
```

Description

The **somfAddMMB** method adds the stipulated key/value pair to the hash table represented by the receiving object, unless the boolean argument prohibits replacement of a conflicting (key, value) pair.

If replace = TRUE, the (key, value) pair is added to the hash table regardless of whether a copy (that is, a pair having the same key) already exists. Otherwise, if *replace* = FALSE, then the (key, value) pair is added to the hash table only if a copy does not exist.

Using the specified *key* and *value* arguments, this method transparently creates an object of the **somf_TAssoc** class, before adding the new (key, value) pair to the hash table.

Note: You cannot override this method.

Parameters

receiver	A pointer to an object of class **somf_THashTable.**
ev	A pointer to the **Environment** structure for the calling method.
key	A pointer to a **somf_MCollectible** object that is the key of the associated pair.
value	A pointer to a **somf_MCollectible** object that is the value of the associated pair.
replace	A boolean indicating whether an already existing pair with an identical key should be replaced.

Return Value

There are two possible valid return values for this method:

somf_MCollectible
: A pointer to the **somf_MCollectible** value that had to be removed in order to add a new (key, value) pair.

SOMF_NIL
: No **somf_MCollectible** value had to be removed in order to add the pair.

Example

```
somf_THashTable ht;
<Your Class which inherits from somf_MCollectible> obj2;
<Your Class which inherits from somf_MCollectible> obj3;
Environment *ev;

ev = somGetGlobalEnvironment();

obj2 = <Your Class which inherits from somf_MCollectible>New();
obj3 = <Your Class which inherits from somf_MCollectible>New();
ht = somf_THashTableNew();
```

```
if (_somfAddMMB(ht, ev, obj2, obj3, FALSE) != SOMF_NIL)
        somPrintf("\n problem adding obj2,obj3 to ht\n");

_somFree (ht);
_somFree (obj2);
_somFree (obj3);
```

Original Class

somf_THashTable

Related Information

Methods: somfAddMM

somfAssign Method

Purpose

Assigns a hash table as being equal to a given source hash table.

IDL Syntax

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

Description

The **somfAssign** method assigns the hash-table receiving object to be "equal" to the source hash table object. 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_THashTable** is used with any other main collection class, then the name of the method must be fully qualified (for example: **somf_THashTable_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, obj);
```

Parameters

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

Return Value

None.

Example

```
somf_THashTable h1;
somf_THashTable h2;
Environment *ev;

ev = somGetGlobalEnvironment();
h1 = somf_THashTableNew();
h2 = somf_THashTableNew();

/* Add a lot of objects to h1 */

/* Assign h2 to the contents of h1 */
somf_THashTable_somfAssign(h2,ev,h1);

_somFree (h1);
_somFree (h2);
```

Original Class

somf_THashTable

somfCount Method

Purpose

Gets the number of objects in the hash table.

IDL Syntax

long somfCount ( );

Description

The **somfCount** method determines the number of objects in the hash table represented by the receiving object, and returns the count as a long.

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 (example: **somf_THashTable_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);
```

Note: You cannot override this method.

Parameters

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

Return Value

This method returns the number of objects in the hash table.

Example

```
somf_THashTable ht;
Environment *ev;

ev = somGetGlobalEnvironment();

ht = somf_THashTableNew();

/* Add some objects to ht */

/* Print the number of objects in ht */
somPrintf("\n Count of ht= %d\n", somf_THashTable_somfCount(ht,ev));

_somFree (ht);
```

Original Class

somf_THashTable

somfDelete Method

Purpose

Deletes a given key and removes the associated (key, value) pair from a hash table, returning a pointer to the value that was removed.

IDL Syntax

somf_MCollectible somfDelete (in somf_MCollectible *key*);

Description

The **somfDelete** method deletes the specified key, and removes the corresponding (key, value) pair from the hash table. The method returns a pointer to the value from the (key, value) pair.

Warning: *Be careful* with **somfDelete.** A hash table does not contain copies of the objects; it contains *pointers* to the objects. Using **somfDelete** deletes the *original object*.

Warning: If an attempt is made to delete a **somf_MCollectible** object that has already been deleted, this will cause a segmentation violation.

Note: You cannot override this method.

Parameters

receiver	A pointer to an object of class **somf_THashTable.**
ev	A pointer to the **Environment** structure for the calling method.
key	A pointer to a **somf_MCollectible** object that is the key of the (key, value) pair to be deleted.

Return Value

There are two possible valid return values for this method:

somf_MCollectible
A pointer to the value that was removed because the key was deleted.

SOMF_NIL The key object was not found.

Example

```
somf_THashTable ht;
<Your Class which inherits from somf_MCollectible> obj;
Environment *ev;

ev = somGetGlobalEnvironment();

obj = <Your Class which inherits from somf_MCollectible>New();
ht = somf_THashTableNew();

/* Add a lot of objects to ht */

/* Remove all occurrences of obj from ht AND DELETE obj */
if (_somfDelete(ht, ev, obj) == SOMF_NIL)
    somPrintf(" Why did Delete say obj wasn't in ht? \n");

_somFree (ht);
```

Original Class

somf_THashTable

Related Information

Methods: somfDeleteAll, somfDeleteAllKeys, somfDeleteAllValues

somfDeleteAll Method

Purpose

Removes all of the (key, value) pairs from a hash table and deallocates the storage that these pairs might have owned. (That is, the destructor function is called for each object in the hash table).

IDL Syntax

void somfDeleteAll ( );

Description

The **somfDeleteAll** method removes all of the (key, value) pairs from the hash table represented by the receiving object. It also deallocates the storage that these pairs might have owned (that is, the destructor function is called for each object in the hash table).

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 a child of **somf_THashTable**, then the name of the method will have to be fully qualified (example: **somf_THashTable_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);
```

Note: You cannot override this method.

Parameters

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

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

Return Value

None.

Example

```
somf_THashTable ht;
Environment *ev;

ev = somGetGlobalEnvironment();

ht = somf_THashTableNew();

/* Add a lot of objects to ht */

/* Remove all objects from ht AND DELETE THEM */
somf_THashTable_somfDeleteAll(ht,ev);

_somFree (ht);
```

Original Class

somf_THashTable

Related Information

Methods: somfDelete, somfDeleteAllKeys, somfDeleteAllValues

somfDeleteAllKeys Method

Purpose

Removes all of the (key, value) pairs from a hash-table receiving object. However, the program still owns the values of the (key, value) pairs.

IDL Syntax

void somfDeleteAllKeys ( );

Description

The **somfDeleteAllKeys** method removes all of the (key, value) pairs from the hash table, deallocates the storage that these objects might have owned, and resets the count to zero. (That is, the destructor function is called for each key in the hash table.) However, the program still owns the objects representing the values of the (key, value) pairs.

Warning: *Be careful* with **somfDeleteAllKeys.** A hash table does not contain copies of the objects; it contains *pointers* to the objects. Using **somfDeleteAllKeys** deletes the *original object*.

Warning: If an attempt is made to delete a **somf_MCollectible** object that has already been deleted, this will cause a segmentation violation.

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 (example: **somf_THashTable_somfDeleteAllKeys**). 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->somfDeleteAllKeys(ev);
```

Note: You cannot override this method.

Parameters

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

Return Value

None.

Example

```
somf_THashTable ht;
Environment *ev;

ev = somGetGlobalEnvironment();

ht = somf_THashTableNew();

/* Add a lot of objects to ht */

/* Remove all objects from ht AND DELETE ALL THE KEYS */
somf_THashTable_somfDeleteAllKeys(ht,ev);

_somFree (ht);
```

Original Class

somf_THashTable

Related Information

Methods: somfDeleteAll, somfDelete, somfDeleteAllValues

somfDeleteAllValues Method

Purpose

Removes all of the (key, value) pairs from a hash table. However, the program still owns the keys of the (key, value) pairs.

IDL Syntax

void somfDeleteAllValues ( );

Description

The **somfDeleteAllValues** method removes all of the (key, value) pairs from the hash table represented by the receiving object. It also deallocates the storage that these objects might have owned and resets the count to zero (that is, the destructor function is called for each value in the hash table.) However, the program still owns the objects representing the keys of the (key, value) pairs.

Warning: *Be careful* with **somfDeleteAllValues.** A hash table does not contain copies of the objects; it contains *pointers* to the objects. Using **somfDeleteAllValues** deletes the *original object*.

Warning: If an attempt is made to delete a **somf_MCollectible** object that has already been deleted, this will cause a segmentation violation.

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 (example: **somf_THashTable_somfDeleteAllValues**). 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->somfDeleteAllValues(ev);
```

Note: You cannot override this method.

Parameters

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

Return Value

None.

Example

```
somf_THashTable ht;
Environment *ev;

ev = somGetGlobalEnvironment();
ht = somf_THashTableNew();

/* Add a lot of objects to ht */

/* Remove all objects from ht AND DELETE ALL THE VALUES */
somf_THashTable_somfDeleteAllValues(ht,ev);

_somFree (ht);
```

Original Class

somf_THashTable

Related Information

Methods: somfDeleteAll, somfDeleteAllKeys, somfDelete

somfGetGrowthRate Method

Purpose

Gets the growth rate of a given hash table.

IDL Syntax

long somfGetGrowthRate ( );

Description

The **somfGetGrowthRate** method returns the growth rate of the hash table represented by the receiving object.

Note: You cannot override this method.

Parameters

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

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

Return Value

This method returns the growth rate for the hash table.

Example

```
somf_THashTable ht;
Environment *ev;

ev = somGetGlobalEnvironment();

ht = somf_THashTableNew();

somPrintf(" Rate should be 20 and is %d \n",
  _somfGetGrowthRate(ht,ev);

_somFree (ht);
```

Original Class

somf_THashTable

Related Information

Methods: somfSetGrowthRate

somfGetHashFunction Method

Purpose

Gets the hash function used by a given hash table.

IDL Syntax

somf_MCollectibleHashFn somfGetHashFunction ( );

Description

The **somfGetHashFunction** method returns the hash function used by the hash table 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 a child of **somf_THashTable** is used with a child of **somf_TDictionary** or **somf_TSet**, then the name of the method will have to be fully qualified (example: **somf_THashTable_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);
```

Note: You cannot override this method.

Parameters

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

Return Value

This method returns a pointer to the hash function.

Example

```
somf_THashTable ht;
Environment *ev;

ev = somGetGlobalEnvironment();

ht = somf_THashTableNew();

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

_somFree (ht);
```

Original Class

somf_THashTable

Related Information

Methods: somfSetHashFunction

somfGetRehashThreshold Method

Purpose

Gets the rehash threshold of a given hash table.

IDL Syntax

long somfGetRehashThreshold ( );

Description

The **somfGetRehashThreshold** method returns the rehash threshold of the hash table represented by the receiving object. The rehash threshold is the percentage of how full the hash table should be before it needs to grow. For example: 80 means 80% of the hash table should be full before the table needs to grow.

Note: You cannot override this method.

Parameters

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

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

Return Value

This method returns the rehash threshold of the hash table.

Example

```
somf_THashTable ht;
Environment *ev;

ev = somGetGlobalEnvironment();

ht = somf_THashTableNew();

somPrintf(" RehashThreshold should be 80 and is %d \n",
           _somfGetRehashThreshold(ht,ev));

_somFree (ht);
```

Original Class

somf_THashTable

Related Information

Methods: somfSetRehashThreshold

somfGrow Method

Purpose

Grows a given hash table.

IDL Syntax

void somfGrow ( );

Description

The **somfGrow** method increases the size allocation for the hash table represented by the receiving object. Growth is determined by the growth rate argument (if any) specified in the initialization method for the hash table, or by the growth rate in the **somfSetGrowthRate** method.

Note: You cannot override this method.

Parameters

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

Return Value

None.

Example

```
somf_THashTable ht;
Environment *ev;

ev = somGetGlobalEnvironment();
ht = somf_THashTableNew();

/* Add a lot of objects to ht */

_somfGrow(ht,ev);

_somFree (ht);
```

Original Class

somf_THashTable

somfMember Method

Purpose

Gets the key of a (key, value) pair in a hash table, if it is found.

IDL Syntax

somf_MCollectible somfMember (in somf_MCollectible *key***);**

Description

The **somfMember** method determines whether the (key, value) pair corresponding to a specified *key* is in the hash table represented by the receiving object and, if so, returns a pointer to the key 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 (example: **somf_THashTable_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, key);
```

Note: You cannot override this method.

Parameters

receiver	A pointer to an object of class **somf_THashTable.**
ev	A pointer to the **Environment** structure for the calling method.
key	A pointer to the **somf_MCollectible** key of the (key, value) pair that may or may not be a member of the Hash Table.

Return Value

There are two possible valid return values for this method:

somf_MCollectible
: A pointer to the key of the (key, value) pair the method determined as the member.

SOMF_NIL
: The object was not found.

Example

```
somf_THashTable ht;
<Your Class which inherits from somf_MCollectible> obj;
Environment *ev;

ev = somGetGlobalEnvironment();

obj = <Your Class which inherits from somf_MCollectible>New();
ht = somf_THashTableNew();

/* Add a lot of objects to ht */

/* See if obj is in ht */
if (somf_THashTable_somfMember(ht, ev,obj) != SOMF_NIL)
    somPrintf("\n obj IS in ht\n");
else
    somPrintf("\n obj is NOT in ht\n");

_somFree (ht);
```

Original Class

somf_THashTable

somfRemove Method

Purpose

Removes from the hash table the (key, value) pair associated with a given key.

IDL Syntax

somf_MCollectible somfRemove (in somf_MCollectible *key***);**

Description

The **somfRemove** method removes from the hash table the (key, value) pair associated with the specified *key* object, and returns a pointer to the value that was removed.

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_THashTable_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, key);
```

Note: You cannot override this method.

Parameters

receiver	A pointer to an object of class **somf_THashTable.**
ev	A pointer to the **Environment** structure for the calling method.
key	A pointer to the **somf_MCollectible** key of the (key, value) pair to be removed.

Return Value

There are two possible valid return values for this method:

somf_MCollectible
: A pointer to the value of the (key, value) pair that was removed.

SOMF_NIL
: The *key* object was not found.

Example

```
somf_THashTable ht;
<your Class which inherits from somf_MCollectible> key;
Environment *ev;

ev = somGetGlobalEnvironment();

ht = somf_THashTableNew();
key = <your Class which inherits from somf_MCollectible>New();

/* Add a lot of objects to ht */

if (somf_THashTable_somfRemove(ht, ev, key) == SOMF_NIL)
    somPrintf(" Why did Remove say key was not removed?\n");

_somFree (ht);
_somFree (key);
```

Original Class

somf_THashTable

Related Information

Methods: somfRemoveAll

somfRemoveAll Method

Purpose

Removes all of the (key, value) pairs from a hash table.

IDL Syntax

void somfRemoveAll ( );

Description

The **somfRemoveAll** method removes all of the (key, value) pairs from the hash table that is 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 (example: **somf_THashTable_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);
```

Note: You cannot override this method.

Parameters

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

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

Return Value

None.

Example

```
somf_THashTable ht;
Environment *ev;

ev = somGetGlobalEnvironment();
ht = somf_THashTableNew();

/* Add a lot of objects to ht */

/* Remove all the objects from ht */
somf_THashTable_somfRemoveAll(ht,ev);

_somFree (ht);
```

Original Class

somf_THashTable

Related Information

Methods: somfRemove

somfRetrieve Method

Purpose

Retrieves the value associated with a given key for a (key, value) pair in a hash table.

IDL Syntax

somf_MCollectible somfRetrieve (in somf_MCollectible *key***);**

Description

The **somfRetrieve** method finds the value associated with the specified key for a (key, value) pair in a hash table, and returns a pointer to the value.

Note: You cannot override this method.

Parameters

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

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

key A pointer to the **somf_MCollectible** key for the associated value to be retrieved.

Return Value

There are two possible valid return values for this method:

somf_MCollectible
A pointer to the value associated with the given key.

SOMF_NIL The key was not found in the hash table.

Example

```
somf_THashTable ht;
<Your Class which inherits from somf_MCollectible> key;
<Your Class which inherits from somf_MCollectible> value;
Environment *ev;

ev = somGetGlobalEnvironment();

key = <Your Class which inherits from somf_MCollectible>New();
ht = somf_THashTableNew();

/* Add some objects to ht */

/* Determine the value associated with key */
value = _somfRetrieve(ht, ev, key);

_somFree (ht);
_somFree (key);
```

Original Class

somf_THashTable

somfSetGrowthRate Method

Purpose

Sets the growth rate of a hash table.

IDL Syntax

void somfSetGrowthRate (in long *rate***);**

Description

The **somfSetGrowthRate** method sets the growth rate of the hash table represented by the receiving object.

Note: You cannot override this method.

Parameters

receiver	A pointer to an object of class **somf_THashTable.**
ev	A pointer to the **Environment** structure for the calling method.
rate	The growth rate, expressed as the number of (key, value) pairs by which to expand the table size when it grows.

Return Value

None.

Example

```
somf_THashTable ht;
Environment *ev;

ev = somGetGlobalEnvironment();

ht = somf_THashTableNew();

_somfSetGrowthRate(ht, ev, 20);

_somFree (ht);
```

Original Class

somf_THashTable

Related Information

Methods: somfGetGrowthRate

somfSetHashFunction Method

Purpose

Sets a hash table's hash function to a given function.

IDL Syntax

void somfSetHashFunction (in somf_MCollectibleHashFn *fn*);

Description

The **somfSetHashFunction** method sets the pointer for the hash table's hash function to the specified method *fn*. By default, this pointer is set to **somf_MCollectible**'s **somfHash** method (which is usually overridden in the objects that are added to the hash table). Normally, a client program does not invoke this method.

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_THashTable** is used with a child of **somf_TDictionary** or **somf_TSet**, then the name of the method will have to be fully qualified (example: **somf_THashTable_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, fn);
```

Note: You cannot override this method.

Parameters

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

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_THashTable** 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_THashTable** object.

Return Value

None.

Example

```
somf_THashTable ht;
Environment *ev;

ev = somGetGlobalEnvironment();

ht = somf_THashTableNew();

somf_THashTable_somfSetHashFunction(ht, ev,

somf_MCollectibleClassData.somfHash);

_somFree (ht);
```

Original Class

somf_THashTable

Related Information

Methods: somfGetHashFunction

somfSetRehashThreshold Method

Purpose

Sets the rehash threshold of a hash table.

IDL Syntax

void somfSetRehashThreshold (in long *threshold***);**

Description

The **somfSetRehashThreshold** method sets the rehash threshold of the hash table represented by the receiving object.

Note: You cannot override this method.

Parameters

receiver	A pointer to an object of class **somf_THashTable.**
ev	A pointer to the **Environment** structure for the calling method.
threshold	The rehash threshold, expressed as the percentage of how full the hash table may become before it grows in size. For example: 80 means 80%.

Return Value

None.

Example

```
somf_THashTable ht;
Environment *ev;

ev = somGetGlobalEnvironment();

ht = somf_THashTableNew();

_somfSetRehashThreshold(ht, ev, 80);

_somFree (ht);
```

Original Class

somf_THashTable

Related Information

Methods: somfGetRehashThreshold

somfTHashTableInitFL Method

Purpose

Initializes a new hash table, given its comparison test method and its initial table size.

IDL Syntax

somf_THashTable somfTHashTableInitFL (
in somf_MCollectibleCompareFn *testfn***,**
in long *tablesize***);**

Description

The **somfTHashTableInitFL** method initializes a new hash table, given its comparison test method and its initial table size.

Note: You cannot override this method.

Parameters

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

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

testfn A method pointer specifying either a **somfIsEqual** or **somfIsSame** method. This method is used to compare two keys in the hash table.

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_THashTable** 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_THashTable** object.

tablesize The initial size of the hash table, expressed as the number of (key, value) pairs that are expected.

Return Value

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

Example

```
somf_THashTable h3;
Environment *ev;

ev = somGetGlobalEnvironment();

h3 = somf_THashTableNew();
_somfTHashTableInitFL(h3, ev,
                      somf_MCollectibleClassData.somfIsEqual, 23);

_somFree (h3);
```

Original Class

somf_THashTable

Related Information

Methods: somfTHashTableInitFLLL, somfTHashTableInitFLL, somfTHashTableInitH

somfTHashTableInitFLL Method

Purpose

Initializes a new hash table, given its comparison test method, its initial table size, and its initial growth rate.

IDL Syntax

somf_THashTable somfTHashTableInitFLL (
in somf_MCollectibleCompareFn *testfn***,**
in long *tablesize***,**
in long *rate***);**

Description

The **somfTHashTableInitFLL** method initializes a new hash table, given its comparison test method, its initial table size, and its initial growth rate.

Note: You cannot override this method.

Parameters

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

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

testfn A method pointer specifying either a **somfIsEqual** or **somfIsSame** method. This method is used to compare two keys in the hash table.

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_THashTable** 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_THashTable** object.

tablesize The initial size of the hash table, expressed as the number of (key, value) pairs that are expected.

rate The growth rate, expressed as the number of (key, value) pairs by which to expand the table size when it grows.

Return Value

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

Example

```
somf_THashTable h2;
Environment *ev;

ev = somGetGlobalEnvironment();

h2 = somf_THashTableNew();
_somfTHashTableInitFLL(h2, ev,
                       somf_MCollectibleClassData.somfIsEqual, 23, 20);

_somFree (h2);
```

Original Class

somf_THashTable

Related Information

Methods: somfTHashTableInitFLLL, somfTHashTableInitFL, somfTHashTableInitH

somfTHashTableInitFLLL Method

Purpose

Initializes a new hash table, given its comparison test method, its initial table size, its initial growth rate, and its rehash threshold.

IDL Syntax

somf_THashTable somfTHashTableInitFLLL (
in somf_MCollectibleCompareFn *testfn***,**
in long *tablesize***,**
in long *rate***,**
in long *threshold***);**

Description

The **somfTHashTableInitFLLL** method initializes a new hash table, given its comparison test method, its initial table size, its initial growth rate, and its rehash threshold.

Note: You cannot override this method.

Parameters

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

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

testfn A method pointer specifying either a **somfIsEqual** or **somfIsSame** method. This method is used to compare two keys in the hash table.

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_THashTable** 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_THashTable** object.

tablesize The initial size of the hash table, expressed as the number of (key, value) pairs that are expected.

rate The growth rate, expressed as the number of (key, value) pairs by which to expand the table size when it grows.

threshold The rehash threshold, expressed as the percentage of how full the hash table may become before it grows in size.

Return Value

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

Example

```
somf_THashTable h1;
Environment *ev;

ev = somGetGlobalEnvironment();

h1 = somf_THashTableNew();
_somfTHashTableInitFLLL(h1, ev,
                somf_MCollectibleClassData.somfIsEqual, 23, 20, 80);

_somFree (h1);
```

Original Class

somf_THashTable

Related Information

Methods: somfTHashTableInitFLL, somfTHashTableInitFL, somfTHashTableInitH

somfTHashTableInitH Method

Purpose

Initializes a new hash table, setting it equal to another specified hash table.

IDL Syntax

somf_THashTable somfTHashTableInitH (in somf_THashTable *h*);

Description

The **somfTHashTableInitH** method initializes the new hash table represented by the receiving object. The method also sets the new hash table equal to another specified hash table. This implies that the instance data of the new hash table will be set equal to those of the source hash table.

Note: You cannot override this method.

Parameters

receiver	A pointer to an object of class **somf_THashTable.**
ev	A pointer to the **Environment** structure for the calling method.
h	A pointer to the hash table the receiving object will be equal to.

Return Value

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

Example

```
somf_THashTable h4;
somf_THashTable h2;
Environment *ev;

ev = somGetGlobalEnvironment();

h2 = somf_THashTableNew();
h4 = somf_THashTableNew();
_somfTHashTableInitH(h4, ev, h2);

_somFree (h2);
_somFree (h4);
```

Original Class

somf_THashTable

Related Information

Methods: somfTHashTableInitFLLL, somfTHashTableInitFLL, somfTHashTableInitFL

somf_THashTableIterator Class

Description

The **somf_THashTableIterator** class defines an iterator for the **somf_THashTable** class that will iterate over all of the objects in a hash table.

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. Recall that each entry in a **somf_THashTable** is actually an object of the **somf_TAssoc** class that holds a (key, value) pair. Thus, the **somfFirst** and **somfNext** methods in the **somf_THashTableIterator** class actually return **somf_TAssoc** objects, not simply objects of the **somf_MCollectible** class. You must handle the return values as if they were **somf_TAssoc**'s.

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 is to be passed to multiple threads, the code in those threads must guarantee thread-safe usage of the class.

File Stem

thashitr

Base

somf_TIterator

Metaclass

SOMClass

Ancestor Classes

somf_TIterator, SOMObject

New Methods

somfTHashTableIteratorInit

Overriding Methods

somInit
somUninit
somfFirst
somfNext
somfRemove

somfFirst Method

Purpose

Resets the iterator and returns the first (key, value) pair of a hash table.

IDL Syntax

somf_MCollectible somfFirst ( );

Description

The **somfFirst** method resets the iterator and returns the first (key, value) pair in the hash table that corresponds to the specified hash-table iterator.

This resets the iterator to the beginning of the hash table. This is true not only for the first time you use the iterator; it is also true if other operations on the hash table cause the iterator to be invalidated. In the second case, the method also revalidates the iterator.

Do not be misled by this method's interface, which is inherited from the **somf_TIterator** class. The *only* objects returned with **somfFirst** are (key, value) pairs of the **somf_TAssoc** class. You cannot use the return value as a generic **somf_MCollectible** 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_THashTableIterator_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_THashTableIterator.**
ev	A pointer to the **Environment** structure for the calling method.

Return Value

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

Example

```
somf_THashTable ht;
Environment *ev;
somf_THashTableIterator itr;
somf_TAssoc itrobj;
somf_MCollectible objk;
somf_MCollectible objv;

ev = somGetGlobalEnvironment();

ht = somf_THashTableNew();
itr = somf_THashTableIteratorNew();
_somfTHashTableIteratorInit(itr, ev, ht);

/* Add some object to d */
```

```
/* Iterate through the THashTable */
itrobj = somf_THashTableIterator_somfFirst(itr,ev);
while (itrobj != SOMF_NIL)
{
    objk = _somfGetKey(itrobj,ev);
    objv = _somfGetValue(itrobj,ev);

   /* Do something with objk or objv */

    itrobj = _somfNext(itr,ev);
}

_somFree (ht);
_somFree (itr);
```

Original Class

somf_TIterator (overridden here)

Related Information

Methods: somfNext

somfNext Method

Purpose

Gets the next (key, value) pair from the hash table of a given hash-table iterator.

IDL Syntax

somf_MCollectible somfNext ( );

Description

The **somfNext** method determines the next (key, value) pair in the hash table of the specified hash table iterator. The method also returns a pointer to the next (key, value) pair, if found. Objects are retrieved in an order that reflects the "ordered-ness" of the hash table (or the lack of ordering on the hash table objects).

Do not be misled by this method's interface, which is inherited from the **somf_TIterator** class. The *only* objects returned with **somfNext** are (key, value) pairs of the **somf_TAssoc** class. You cannot use the return value as a generic **somf_MCollectible** object.

If the **somf_THashTable** 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 **somfAdd** method were called after starting to iterate through the hash table, 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_THashTableIterator_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_THashTableIterator.**
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

```
somf_THashTable ht;
Environment *ev;
somf_THashTableIterator itr;
somf_TAssoc itrobj;
somf_MCollectible objk;
somf_MCollectible objv;

ev = somGetGlobalEnvironment();
```

```
ht = somf_THashTableNew();
itr = somf_THashTableIteratorNew();
_somfTHashTableIteratorInit(itr, ev, ht);

/* Add some object to d */

/* Iterate through the THashTable */
itrobj = somf_THashTableIterator_somfFirst(itr,ev);
while (itrobj != SOMF_NIL)
{
    objk = _somfGetKey(itrobj,ev);
    objv = _somfGetValue(itrobj,ev);

   /* Do something with objk or objv */

    itrobj = _somfNext(itr,ev);
}

_somFree (ht);
_somFree (itr);
```

Original Class

somf_TIterator (overridden here)

Related Information

Methods: somfFirst

somfRemove Method

Purpose

Removes the current (key, value) pair (the one just returned by **somfFirst** or **somfNext**) from the hash table.

IDL Syntax

void somfRemove ( );

Description

The **somfRemove** method removes the current (key, value) pair (the object just returned by **somfFirst** or **somfNext**) from the hash table that corresponds to the hash table iterator represented by the receiving object.

The **somfRemove** method is the only way to remove a (key, value) object from a hash table during iteration. However, if multiple iterators are in process, all the other iterators are invalidated, just as if some other kind of change had occurred in the hash table.

If the hash table 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_THashTableIterator_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_THashTableIterator.**
ev	A pointer to the **Environment** structure for the calling method.

Return Value

None.

Example

```
somf_THashTable ht;
Environment *ev;
somf_THashTableIterator itr;
somf_MCollectible itrobj;

ev = somGetGlobalEnvironment();

ht = somf_THashTableNew();
itr = somf_THashTableIteratorNew();
_somfTHashTableIteratorInit(itr, ev, ht);

/* Add some objects to ht */

/* Use the Iterator's Remove to remove the first object */
itrobj = somf_THashTableIterator_somfFirst(itr,ev);
somf_THashTableIterator_somfRemove(itr,ev);
```

```
_somFree (ht);
_somFree (itr);
```

Original Class

somf_TIterator (overridden here)

somfTHashTableIteratorInit Method

Purpose

Initializes a **somf_THashTableIterator** iterator, given its corresponding hash table.

IDL Syntax

somf_THashTableIterator somfTHashTableIteratorInit (in somf_THashTable *h*);

Description

The **somfTHashTableIteratorInit** method initializes a **somf_THashTableIterator** iterator, given the **somf_THashTable** hash table over which iteration is needed.

Note: This is the only way to initialize a **somf_THashTableIterator** iterator to point to an instance of a **somf_THashTable** object.

Note: You cannot override this method.

Parameters

receiver	A pointer to an object of class **somf_THashTableIterator.**
ev	A pointer to the **Environment** structure for the calling method.
h	A pointer to the hash table the receiving object will iterate over.

Return Value

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

Example

```
somf_THashTable ht;
Environment *ev;
somf_THashTableIterator itr;

ev = somGetGlobalEnvironment();

ht = somf_THashTableNew();
itr = somf_THashTableIteratorNew();
_somfTHashTableIteratorInit(itr, ev, ht);

_somFree (ht);
_somFree (itr);
```

Original Class

somf_THashTableIterator

somf_TIterator Class

Description

Each of the main collection classes has a corresponding iterator class. An iterator for a particular collection object (data structure) will iterate over all of the objects contained therein. The **somf_TIterator** class is the abstract base class for all iterator classes, defining the generic methods used for iteration.

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

If you create classes that inherit from the **somf_TIterator** class, the new classes *must* override the methods **somfFirst** and **somfNext**.

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

File Stem

titeratr

Base

SOMObject

Metaclass

SOMClass

Ancestor Classes

SOMObject

New Methods

somfNext
somfFirst
somfRemove

Overriding Methods

None

somfFirst Method

Purpose

Resets the iterator and returns the first object of a collection.

IDL Syntax

somf_MCollectible somfFirst ( );

Description

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

This resets the iterator to the beginning of the collection. This is true not only for the first time you use the iterator; 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 **somf_TIterator** *must* override this method for the 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_TDictionaryIterator_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_TIterator.**
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 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 looks when it is invoked, see **somf_TSetIterator** or **somf_TDictionaryIterator**, or any of the other classes that inherit from **somf_TIterator**.

Original Class

somf_TIterator

Related Information

Methods: somfNext

somfNext Method

Purpose

Gets the next object in a collection.

IDL Syntax

somf_MCollectible somfNext ( );

Description

The **somfNext** method determines the next object in the collection that corresponds to the iterator represented by the receiving object and, if found, returns a pointer to the object. Objects are retrieved in an order that reflects the "ordered-ness" of the collection (or the lack of ordering on the collection objects).

Every class that inherits from this class *must* override this method for the 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 (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_TIterator.**
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 looks when it is invoked, see **somf_TSetIterator** or **somf_TDictionaryIterator**, or any of the other classes that inherit from **somf_TIterator**.

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 collection.

IDL Syntax

void somfRemove ( );

Description

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

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

This method is the only way to remove an object from a 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 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 (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_TIterator.**
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 looks when it is invoked, see **somf_TSetIterator** or **somf_TDictionaryIterator**, or any of the other classes that inherit from **somf_TIterator**.

Original Class

somf_TIterator

somf_TPrimitiveLinkedList Class

Description

This class describes a primitive linked list — a sequence of zero or more items, each linked to the item in front and to the item behind it.

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

Objects that are inserted into a collection object of the **somf_TPrimitiveLinkedList** class *must* inherit from the **somf_MLinkable** class.

Warning: The **somf_TPrimitiveLinkedList** class uses the left and right pointers of the **somf_MLinkable** characteristics to link together the objects in a list. This means no object can appear in the list more than once, since it only has one set of pointers to indicate its position in the **somf_TPrimitiveLinkedList**. If you insert an object more than once, the behavior is undefined, and could result in an infinite loop. If you need to insert an object more than once, you should consider using a **somf_TDeque** collection instead. For the same reasons, an item cannot appear in two different linked lists, because the same undefined behavior would result.

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

tpll

Base

SOMObject

Metaclass

SOMClass

Ancestor Classes

SOMObject

New Methods

somfCount
somfRemove
somfRemoveAll,
somfRemoveFirst
somfRemoveLast
somfAddBefore
somfAddAfter
somfAddFirst
somfAddLast
somfAfter
somfBefore
somfFirst
somfLast

Overriding Methods

somInit
somUninit

somfAddAfter Method

Purpose

Adds an object into a list after a given existing object.

IDL Syntax

void somfAddAfter (
in somf_MLinkable *existing*,
in somf_MLinkable *obj*);

Description

The **somfAddAfter** method adds the object `obj` into the specified list after the designated existing object.

Parameters

receiver	A pointer to an object of class **somf_TPrimitiveLinkedList.**
ev	A pointer to the **Environment** structure for the calling method.
existing	A pointer to the **somf_MLinkable** object that *obj* will be added after.
obj	A pointer to the **somf_MLinkable** object that will be added.

Return Value

None.

Example

```
somf_TPrimitiveLinkedList l;
<Your Class which inherits from MLinkable> obj;
<Your Class which inherits from MLinkable> obj2;
Environment *ev;

ev = somGetGlobalEnvironment();

l = somf_TPrimitiveLinkedListNew();
obj = <Your Class which inherits from MLinkable>New();
obj2 = <Your Class which inherits from MLinkable>New();

/* Add obj2 to l after obj */
_somfAddFirst(l, ev, obj);
_somfAddAfter(l, ev, obj, obj2);

_somFree (l);
_somFree (obj);
_somFree (obj2);
```

Original Class

somf_TPrimitiveLinkedList

Related Information

Methods: somfAddBefore, somfAddFirst, somfAddLast

somfAddBefore Method

Purpose

Adds an object into a list before a given existing object.

IDL Syntax

void somfAddBefore (
in somf_MLinkable *existing*,
in somf_MLinkable *obj*);

Description

The **somfAddBefore** method adds the object `obj` into the specified list before the designated existing object.

Parameters

receiver	A pointer to an object of class **somf_TPrimitiveLinkedList.**
ev	A pointer to the **Environment** structure for the calling method.
existing	A pointer to the **somf_MLinkable** object that *obj* will be added in front of.
obj	A pointer to the **somf_MLinkable** object that will be added.

Return Value

None.

Example

```
somf_TPrimitiveLinkedList l;
<Your Class which inherits from MLinkable> obj;
<Your Class which inherits from MLinkable> obj2;
Environment *ev;

ev = somGetGlobalEnvironment();

l = somf_TPrimitiveLinkedListNew();
obj = <Your Class which inherits from MLinkable>New();
obj2 = <Your Class which inherits from MLinkable>New();

/* Add obj2 to l before obj */
_somfAddFirst(l, ev, obj);
_somfAddBefore(l, ev, obj, obj2);

_somFree (l);
_somFree (obj);
_somFree (obj2);
```

Original Class

somf_TPrimitiveLinkedList

Related Information

Methods: somfAddAfter, somfAddFirst, somfAddLast

somfAddFirst Method

Purpose

Adds an object as the first object in a list.

IDL Syntax

void somfAddFirst (in somf_MLinkable *obj***);**

Description

The **somfAddFirst** method adds the object `obj` as the first object in the specified list.

Parameters

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

Return Value

None.

Example

```
somf_TPrimitiveLinkedList l;
<Your Class which inherits from MLinkable> obj;
Environment *ev;

ev = somGetGlobalEnvironment();

l = somf_TPrimitiveLinkedListNew();
obj = <Your Class which inherits from MLinkable>New();

/* Add obj to the front of l */
_somfAddFirst(l, ev, obj);

_somFree (l);
_somFree (obj);
```

Original Class

somf_TPrimitiveLinkedList

Related Information

Methods: somfAddAfter, somfAddBefore, somfAddLast

somfAddLast Method

Purpose

Adds an object as the last object in a given list.

IDL Syntax

void somfAddLast (in somf_MLinkable *obj***);**

Description

The **somfAddLast** method adds the object `obj` as the last object in the specified list.

Parameters

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

Return Value

None.

Example

```
somf_TPrimitiveLinkedList l;
<Your Class which inherits from MLinkable> obj;
Environment *ev;

ev = somGetGlobalEnvironment();

l = somf_TPrimitiveLinkedListNew();
obj = <Your Class which inherits from MLinkable>New();

/* Add obj to the end of l */
_somfAddLast(l, ev, obj);

_somFree (l);
_somFree (obj);
```

Original Class

somf_TPrimitiveLinkedList

Related Information

Methods: somfAddAfter, somfAddBefore, somfAddFirst

somfAfter Method

Purpose

Gets the object that comes after a given existing object in a list.

IDL Syntax

somf_MLinkable somfAfter (in somf_MLinkable *existingobj***);**

Description

The **somfAfter** method returns the object that comes after the object *existingobj* in the specified list.

Parameters

receiver	A pointer to an object of class **somf_TPrimitiveLinkedList.**
ev	A pointer to the **Environment** structure for the calling method.
existingobj	A pointer to the **somf_MLinkable** that is in front of the returned object.

Return Value

There are two possible valid return values for this method:

somf_MLinkable
A pointer to the **somf_MLinkable** object after the *existingobj* object.

SOMF_NIL Nothing is after *existingobj.*

Example

```
somf_TPrimitiveLinkedList l;
<Your Class which inherits from MLinkable> obj;
somf_MLinkable obj2;
Environment *ev;

ev = somGetGlobalEnvironment();

l = somf_TPrimitiveLinkedListNew();
obj = <Your Class which inherits from MLinkable>New();

/* Add a lot of objects to l */

/* Determine the object in l after obj */
obj2 = _somfAfter(l, ev, obj);

_somFree (l);
_somFree (obj);
```

Original Class

somf_TPrimitiveLinkedList

Related Information

Methods: somfBefore

somfBefore Method

Purpose

Returns the object that comes before a given existing object in a list.

IDL Syntax

somf_MLinkable somfBefore (in somf_MLinkable *existingobj***);**

Description

The **somfBefore** method returns the object that comes before the object *existingobj* in the specified list.

Parameters

receiver	A pointer to an object of class **somf_TPrimitiveLinkedList.**
ev	A pointer to the **Environment** structure for the calling method.
existingobj	A pointer to the **somf_MLinkable** object that comes after the returned object.

Return Value

There are two possible valid return values for this method:

somf_MLinkable
A pointer to the **somf_MLinkable** object before the *existingobj* object.

SOMF_NIL Nothing is before the *existingobj.*

Example

```
somf_TPrimitiveLinkedList l;
<Your Class which inherits from MLinkable> obj;
somf_MLinkable obj2;
Environment *ev;

ev = somGetGlobalEnvironment();

l = somf_TPrimitiveLinkedListNew();
obj = <Your Class which inherits from MLinkable>New();

/* Add a lot of objects to l */

/* Determine the object in l before obj */
obj2 = _somfBefore(l, ev, obj);

_somFree (l);
_somFree (obj);
```

Original Class

somf_TPrimitiveLinkedList

Related Information

Methods: somfAfter

somfCount Method

Purpose

Gets the number of objects in a given list.

IDL Syntax

unsigned long somfCount ( );

Description

The **somfCount** method determines the number of objects in the specified list, and returns the number.

Parameters

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

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

Return Value

This method returns the number of objects in the specified list.

Example

```
somf_TPrimitiveLinkedList l;
Environment *ev;

ev = somGetGlobalEnvironment();

l = somf_TPrimitiveLinkedListNew();

/* Add some objects to l */

/* Print the number of objects in ht */
somPrintf("\n Count of l= %d\n",
          somf_TPrimitiveLinkedList_somfCount(l,ev));

_somFree (l);
```

Original Class

somf_TPrimitiveLinkedList

somfFirst Method

Purpose

Gets the first object in a given list.

IDL Syntax

somf_MLinkable somfFirst ( );

Description

The **somfFirst** method returns the first object in the specified list.

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_TPrimitiveLinkedList_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:

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

Parameters

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

Return Value

There are two possible valid return values for this method:

somf_MLinkable
A pointer to the first **somf_MLinkable** object in the list.

SOMF_NIL Nothing is in the list.

Example

```
somf_TPrimitiveLinkedList l;
somf_MLinkable obj;
Environment *ev;

ev = somGetGlobalEnvironment();

l = somf_TPrimitiveLinkedListNew();

/* Add a lot of objects to l */

/* Determine the first object in l */
obj = somf_TPrimitiveLinkedList_somfFirst(l,ev);

_somFree (l);
```

Original Class

somf_TPrimitiveLinkedList

Related Information

Methods: somfLast

somfLast Method

Purpose

Gets the last object in a given list.

IDL Syntax

somf_MLinkable somfLast ( );

Description

The **somfLast** method returns the last object in the specified list.

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_TPrimitiveLinkedList_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:

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

Parameters

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

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

Return Value

There are two possible valid return values for this method:

somf_MLinkable
A pointer to the last **somf_MLinkable** object in the list.

SOMF_NIL Nothing is in the list.

Example

```
somf_TPrimitiveLinkedList l;
somf_MLinkable obj;
Environment *ev;

ev = somGetGlobalEnvironment();

l = somf_TPrimitiveLinkedListNew();

/* Add a lot of objects to l */

/* Determine the last object in l */
obj = somf_TPrimitiveLinkedList_somfLast(l,ev);

_somFree (l);
```

Original Class

somf_TPrimitiveLinkedList

Related Information

Methods: somfFirst

somfRemove Method

Purpose

Removes a **somf_MLinkable** object from a given list.

IDL Syntax

void somfRemove (in somf_MLinkable *aLink***);**

Description

The **somfRemove** method removes the specified **somf_MLinkable** object from the designated list.

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_TPrimitiveLinkedList_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:

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

Parameters

receiver	A pointer to an object of class **somf_TPrimitiveLinkedList.**
ev	A pointer to the **Environment** structure for the calling method.
aLink	A pointer to the **somf_MLinkable** object to be removed.

Return Value

None.

Example

```
somf_TPrimitiveLinkedList l;
<Your Class which inherits from MLinkable> obj;
Environment *ev;

ev = somGetGlobalEnvironment();

l = somf_TPrimitiveLinkedListNew();
obj = <Your Class which inherits from MLinkable>New();

/* Add a lot of objects to l */

/* Remove obj from l */
somf_TPrimitiveLinkedList_somfRemove(l, ev, obj);

_somFree (l);
_somFree (obj);
```

Original Class

somf_TPrimitiveLinkedList

Related Information

Methods: somfRemoveAll, somfRemoveFirst, somfRemoveLast

somfRemoveAll Method

Purpose

Removes all of the objects from a given list.

IDL Syntax

void somfRemoveAll ( );

Description

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

Parameters

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

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

Return Value

None.

Example

```
somf_TPrimitiveLinkedList l;
<Your Class which inherits from MLinkable> obj;
Environment *ev;

ev = somGetGlobalEnvironment();

l = somf_TPrimitiveLinkedListNew();
obj = <Your Class which inherits from MLinkable>New();

/* Add a lot of objects to l */

/* Remove all of the objects from l */
somf_TPrimitiveLinkedList_somfRemoveAll(l,ev);

_somFree (l);
_somFree (obj);
```

Original Class

somf_TPrimitiveLinkedList

Related Information

Methods: somfRemove, somfRemoveFirst, somfRemoveLast

somfRemoveFirst Method

Purpose

Removes the first object from a given list.

IDL Syntax

somf_MLinkable somfRemoveFirst ( );

Description

The **somfRemoveFirst** method removes the first object from the list represented by the receiving object.

Parameters

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

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

Return Value

There are two possible valid return values for this method:

somf_MLinkable
A pointer to the **somf_MLinkable** object removed from the list.

SOMF_NIL Nothing is in the list.

Example

```
somf_TPrimitiveLinkedList l;
Environment *ev;

ev = somGetGlobalEnvironment();

l = somf_TPrimitiveLinkedListNew();

/* Add some objects to l */

/* Remove the first object */
if (_somfRemoveFirst(l,ev) == SOMF_NIL)
    somPrintf(" The list is empty\n");

_somFree (l);
```

Original Class

somf_TPrimitiveLinkedList

Related Information

Methods: somfRemove, somfRemoveAll, somfRemoveLast

somfRemoveLast Method

Purpose

Removes the last object from a given list.

IDL Syntax

somf_MLinkable somfRemoveLast ( );

Description

The **somfRemoveLast** method removes the last object from the list represented by the receiving object.

Parameters

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

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

Return Value

There are two possible valid return values for this method:

somf_MLinkable
A pointer to the **somf_MLinkable** object removed from the list.

SOMF_NIL Nothing is in the list.

Example

```
somf_TPrimitiveLinkedList l;
Environment *ev;

ev = somGetGlobalEnvironment();

l = somf_TPrimitiveLinkedListNew();

/* Add some objects to l */

/* Remove the last object */
if (_somfRemoveLast(l,ev) == SOMF_NIL)
    somPrintf(" The list is empty\n");

_somFree (l);
```

Original Class

somf_TPrimitiveLinkedList

Related Information

Methods: somfRemove, somfRemoveAll, somfRemoveFirst

somf_TPrimitiveLinkedListIterator Class

Description

This class defines an iterator for the **somf_TPrimitiveLinkedList** class that will iterate over all of the objects in a primitive linked list.

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

tpllitr

Base

SOMObject

Metaclass

SOMClass

Ancestor Classes

SOMObject

New Methods

somfFirst
somfNext
somfLast
somfPrevious
somfTPrimitiveLinkedListIteratorInit

Overriding Methods

somUninit

somfFirst Method

Purpose

Resets the iterator and returns the first element of a given list.

IDL Syntax

somf_MLinkable somfFirst ( );

Description

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

Note: The **somf_TPrimitiveLinkedListIterator** class does not inherit from **somf_TIterator**. This method may look like the **somf_TIterator** method, but there is no connection.

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_TPrimitiveLinkedListIterator_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_TPrimitiveLinkedListIterator.**

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

Return Value

There are two possible valid return values for this method:

somf_MLinkable
A pointer to the first **somf_MLinkable** object in the list.

SOMF_NIL Nothing is in the list.

Example

```
somf_TPrimitiveLinkedList l;
somf_MLinkable obj;
somf_TPrimitiveLinkedListIterator itr;
Environment *ev;

ev = somGetGlobalEnvironment();

l = somf_TPrimitiveLinkedListNew();
itr = somf_TPrimitiveLinkedListIteratorNew();
_somfTPrimitiveLinkedListIteratorInit(itr, ev, l);

/* Add a lot of objects to l */
```

```
/* Iterate through l */
obj = somf_TPrimitiveLinkedListIterator_somfFirst(itr,ev);
while (obj != SOMF_NIL)
{
    /* do something with obj */

    obj = _somfNext(itr,ev);
}

_somFree (l);
_somFree (itr);
```

Original Class

somf_TPrimitiveLinkedListIterator

Related Information

Methods: somfNext

somfLast Method

Purpose

Retrieves the last object from a given list.

IDL Syntax

somf_MLinkable somfLast ( );

Description

The **somfLast** method determines the last object in the list that corresponds to the iterator represented by the receiving object and, if found, 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. **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_TPrimitiveLinkedListIterator_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_TPrimitiveLinkedListIterator.**
ev	A pointer to the **Environment** structure for the calling method.

Return Value

There are two possible valid return values for this method:

somf_MLinkable
: A pointer to the last **somf_MLinkable** object in the list.

SOMF_NIL
: Nothing is in the list.

Example

```
somf_TPrimitiveLinkedList l;
somf_MLinkable obj;
somf_TPrimitiveLinkedListIterator itr;
Environment *ev;

ev = somGetGlobalEnvironment();

l = somf_TPrimitiveLinkedListNew();
itr = somf_TPrimitiveLinkedListIteratorNew();
_somfTPrimitiveLinkedListIteratorInit(itr, ev, l);

/* Add a lot of objects to l */

/* Find the last object in l */
obj = somf_TPrimitiveLinkedList_somfLast(l,ev);

_somFree (l);
_somFree (itr);
```

Original Class

somf_TPrimitiveLinkedListIterator

Related Information

Methods: somfPrevious

somfNext Method

Purpose

Gets the next object in a list.

IDL Syntax

somf_MLinkable somfNext ( );

Description

The **somfNext** method determines the next object in the list that corresponds to the iterator represented by the receiving object and, if found, returns a pointer to the object.

Note: The **somf_TPrimitiveLinkedListIterator** class does not inherit from **somf_TIterator**. This method may look like the **somf_TIterator** method, but there is no connection.

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_TPrimitiveLinkedListIterator** is used with **somf_TIterator**, then the name of the method will have to be fully qualified (example: **somf_TPrimitiveLinkedListIterator_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_TPrimitiveLinkedListIterator.**
ev	A pointer to the **Environment** structure for the calling method.

Return Value

There are two possible valid return values for this method:

somf_MLinkable
A pointer to the next **somf_MLinkable** object in the list.

SOMF_NIL The end of the list has been reached.

Example

```
somf_TPrimitiveLinkedList l;
somf_MLinkable obj;
somf_TPrimitiveLinkedListIterator itr;
Environment *ev;

ev = somGetGlobalEnvironment();

l = somf_TPrimitiveLinkedListNew();
itr = somf_TPrimitiveLinkedListIteratorNew();
_somfTPrimitiveLinkedListIteratorInit(itr, ev, l);

/* Add a lot of objects to l */
```

```
/* Iterate through l */
obj = somf_TPrimitiveLinkedListIterator_somfFirst(itr,ev);
while (obj != SOMF_NIL)
{
    /* do something with obj */

    obj = _somfNext(itr,ev);
}

_somFree (l);
_somFree (itr);
```

Original Class

somf_TPrimitiveLinkedListIterator

Related Information

Methods: somfFirst

somfPrevious Method

Purpose

Gets the previous object from a given list.

IDL Syntax

somf_MLinkable somfPrevious ( );

Description

The **somfPrevious** method determines the previous object in the list that corresponds to the iterator 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. Thus, if any child of **somf_TPrimitiveLinkedListIterator** is used with **somf_TSequenceIterator**, then the name of the method will have to be fully qualified (for example: **somf_TPrimitiveLinkedListIterator_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_TPrimitiveLinkedListIterator.**
ev	A pointer to the **Environment** structure for the calling method.

Return Value

There are two possible valid return values for this method:

somf_MLinkable
: A pointer to the **somf_MLinkable** object before the receiving object.

SOMF_NIL
: The beginning of the list has been reached.

Example

```
somf_TPrimitiveLinkedList l;
somf_MLinkable obj;
somf_TPrimitiveLinkedListIterator itr;
Environment *ev;

ev = somGetGlobalEnvironment();

l = somf_TPrimitiveLinkedListNew();
itr = somf_TPrimitiveLinkedListIteratorNew();
_somfTPrimitiveLinkedListIteratorInit(itr, ev, l);

/* Add a lot of objects to l */

/* Find the next to the last object in l */
somf_TPrimitiveLinkedList_somfLast(l,ev);
obj = _somfPrevious(next,ev);

_somFree (l);
_somFree (itr);
```

Original Class

somf_TPrimitiveLinkedListIterator

Related Information

Methods: somfLast

somfTPrimitiveLinkedListIteratorInit Method

Purpose

Initializes a **somf_TPrimitiveLinkedListIterator** object, establishing it as the iterator for a given **somf_TPrimitiveLinkedList** linked list.

IDL Syntax

somf_TPrimitiveLinkedListIterator somfTPrimitiveLinkedListIteratorInit (
in somf_TPrimitiveLinkedList *list***);**

Description

The **somfTPrimitiveLinkedListIteratorInit** method initializes a given iterator object (the **somf_TPrimitiveLinkedListIterator** receiving object) that will iterate over the specified **somf_TPrimitiveLinkedList** list.

Note: You cannot override this method.

Parameters

receiver	A pointer to an object of class **somf_TPrimitiveLinkedListIterator.**
ev	A pointer to the **Environment** structure for the calling method.
list	A pointer to the primitive linked list object that the receiving object will iterate over.

Return Value

This method returns a pointer to an initialized **somf_TPrimitiveLinkedListIterator** iterator.

Example

```
somf_TPrimitiveLinkedList l;
Environment *ev;
somf_TPrimitiveLinkedListIterator itr;

ev = somGetGlobalEnvironment();

l = somf_TPrimitiveLinkedListNew();
itr = somf_TPrimitiveLinkedListIteratorNew();
_somfTPrimitiveLinkedListIteratorInit(itr, ev, l);

_somFree (l);
_somFree (itr);
```

Original Class

somf_TPrimitiveLinkedListIterator

somf_TPriorityQueue Class

Description

The **somf_TPriorityQueue** class is a subclass of **somf_TCollection** that keeps the objects of a collection ordered based on some ordering function. Actually, the objects are partially ordered in storage, but the **somf_TPriorityQueue** methods adjust for the partially ordered state.

Robert Sedgewick[3] describes a Priority Queue as follows:

> In many applications, records with keys must be processed in order, but not necessarily in full sorted order and not necessarily all at once. Often a set of records must be collected, then the largest processed, then perhaps more records collected, then the next largest processed, and so forth. An appropriate data structure in such an environment is one that supports the operations of inserting a new element and deleting the largest element. Such a data structure, which can be contrasted with queues (delete the oldest) and stacks (delete the newest) is called a *priority queue*.

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

Note: The **somf_TPriorityQueue** 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 the initialization methods to change to the **somfIsSame** method.

Objects that are inserted into a **somf_TPriorityQueue** collection should override the methods **somfIsEqual**, **somfIsLessThan**, **somfIsGreaterThan**, and **somfHash**.

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

tpq

Base

somf_TCollection

Metaclass

SOMClass

Ancestor Classes

somf_TCollection, somf_MCollectible, SOMObject

New Methods

somfInsert
somfPop
somfPeek
somfReplace
somfSetEqualityComparisonFunction
somfGetEqualityComparisonFunction
somfAssign
somfTPriorityQueueInitF
somfTPriorityQueueInitP

3. Robert Sedgewick, *Algorithms in C++* (Addison–Wesley Publishing Company, 1992), p. 145.

Overriding Methods

somInit
somUninit
somfAdd
somfRemove
somfRemoveAll
somfDeleteAll
somfCount
somfMember
somfCreateIterator

somfAdd Method

Purpose

Adds a given `obj` to a priority queue.

IDL Syntax

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

Description

The **somfAdd** method adds the specified object *obj* to the priority queue represented by the receiving object.

Parameters

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

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

Example

```
somf_TPriorityQueue pq;
<Your Class which inherits from somf_MOrderableCollectible> obj;
Environment *ev;

ev = somGetGlobalEnvironment();

pq = somf_TPriorityQueueNew();
obj = <Your Class which inherits from somf_MOrderableCollectible>New();

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

_somFree (pq);
_somFree (obj);
```

Original Class

somf_TCollection (overridden here)

Related Information

Methods: somfInsert

somfAssign Method

Purpose

Assigns a priority-queue receiving object as being equal to a given source priority queue.

IDL Syntax

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

Description

The **somfAssign** method assigns the instance of the priority queue used as the receiving object to be equal to the source priority queue. 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_TPriorityQueue** is used with any other main collection class, then the name of the method will have to be fully qualified (example: **somf_TPriorityQueue_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, obj);
```

Parameters

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

Return Value

None.

Example

```
somf_TPriorityQueue pq1;
somf_TPriorityQueue pq2;
Environment *ev;

ev = somGetGlobalEnvironment();

pq1 = somf_TPriorityQueueNew();
pq2 = somf_TPriorityQueueNew();

/* Add some objects to pq1 */

/* Assign pq2 = pq1 */
somf_TPriorityQueue_somfAssign(pq2,ev,pq1);

_somFree (pq1);
_somFree (pq2);
```

Original Class

somf_TPriorityQueue

somfCount Method

Purpose

Gets the number of objects in a given priority queue.

IDL Syntax

long somfCount ( );

Description

The **somfCount** method determines the number of objects in the priority queue represented by the receiving object, and returns the 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 (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_TPriorityQueue.**
ev	A pointer to the **Environment** structure for the calling method.

Return Value

This method returns the number of objects in the receiving object.

Example

```
somf_TPriorityQueue pq;
Environment *ev;

ev = somGetGlobalEnvironment();

pq = somf_TPriorityQueueNew();

/* Add some objects to pq */

/* Count the number of objects in pq */
somPrintf("\n Count of pq= %d\n", _somfCount(pq,ev));

_somFree (pq);
```

Original Class

somf_TCollection (overridden here)

somfCreateIterator Method

Purpose

Returns a new iterator that is suitable for iterating over the objects in a given priority queue.

IDL Syntax

somf_TIterator somfCreateIterator ( );

Description

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

Note: This is one of two ways to initialize a **somf_TPriorityQueueIterator** to point to an instance of the **somf_TPriorityQueue** class. The other way is to use the **somf_TPriorityQueueIterator**'s initializer method described on page 245.

Parameters

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

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

Return Value

This method returns a pointer to the new iterator.

Example

```
somf_TPriorityQueue pq;
Environment *ev;
somf_TPriorityQueueIterator itr;

ev = somGetGlobalEnvironment();

pq = somf_TPriorityQueueNew();
itr = (somf_TPriorityQueueIterator*) _somfCreateIterator(pq,ev);

_somFree (pq);
_somFree (itr);
```

Original Class

somf_TCollection (overridden here)

somfDeleteAll Method

Purpose

Removes all of the objects from a priority-queue receiving object 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 priority queue 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 (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_TPriorityQueue.**
ev	A pointer to the **Environment** structure for the calling method.

Return Value

None.

Example

```
somf_TPriorityQueue pq;
Environment *ev;

ev = somGetGlobalEnvironment();

pq = somf_TPriorityQueueNew();

/* Add objects to pq */

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

_somFree (pq);
```

Original Class

somf_TCollection (overridden here)

somfGetEqualityComparisonFunction Method

Purpose

Gets the equality comparison function being used by the priority queue. The default equality compare function is the **somf_MCollectible** class's **somfIsEqual**.

IDL Syntax

somf_MCollectibleCompareFn somfGetEqualityComparisonFunction ( );

Description

The **somfGetEqualityComparisonFunction** method returns the equality comparison function being used by the priority queue. By default, the equality compare function is the **somf_MCollectible** class's **somfIsEqual** method.

Note: Do not confuse this "equality compare function" with the **somfCompare** method. This input argument is *not* used to determine priority.

Parameters

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

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

Return Value

This method returns a pointer to the equality compare function being used by this instance of the priority queue class.

Example

```
somf_TPriorityQueue pq;
Environment *ev;

ev = somGetGlobalEnvironment();

pq = somf_TPriorityQueueNew();

/* Add some objects to pq */

if (_somfGetEqualityComparisonFunction(pq,ev) !=
    somf_MCollectibleClassData.somfIsEqual)
{
    somPrintf("\n What Compare Function are we using?\n");
}

_somFree (pq);
```

Original Class

somf_TPriorityQueue

Related Information

Methods: somfSetEqualityComparisonFunction

somfInsert Method

Purpose

Inserts an object `obj` into the priority queue.

IDL Syntax

void somfInsert (in somf_MOrderableCollectible *obj*);

Description

The **somfInsert** method inserts the given object *obj* into the priority queue represented by the receiving object.

This method is just like the **somfAdd** method, except that it does not return a pointer to the **somf_MCollectible** object added.

Parameters

receiver	A pointer to an object of class **somf_TPriorityQueue.**
ev	A pointer to the **Environment** structure for the calling method.
obj	A pointer to a **somf_MOrderableCollectible** object that will be added to the receiving object.

Return Value

None.

Example

```
somf_TPriorityQueue pq;
<Your Class which inherits from somf_MOrderableCollectible> obj;
Environment *ev;

ev = somGetGlobalEnvironment();

pq = somf_TPriorityQueueNew();
obj = <Your Class which inherits from somf_MOrderableCollectible>New();

/* Add obj to pq */
_somfInsert(pq,ev,obj);

_somFree (pq);
_somFree (obj);
```

Original Class

somf_TPriorityQueue

Related Information

Methods: somfAdd

somfMember Method

Purpose

Gets an object from a given priority queue.

IDL Syntax

somf_MCollectible somfMember (in somf_MCollectible *obj***);**

Description

The **somfMember** method determines whether a specified object *obj* is in the priority queue represented by the receiving object and, if so, 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 (example: **somf_TDictionary_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_TPriorityQueue.**
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_TPriorityQueue pq;
<your Class which inherits from somf_MOrderableCollectible> obj;
Environment *ev;

ev = somGetGlobalEnvironment();

pq = somf_TPriorityQueueNew();
obj = <your Class which inherits from somf_MOrderableCollectible>New();

/* Add some objects to pq */

/* See if obj is in pq */
if (somf_TPriorityQueue_somfMember(pq, ev, obj) == SOMF_NIL)
    somPrintf("\n obj is NOT in d\n");
else
    somPrintf("\n obj IS in d\n");

_somFree (pq);
```

Original Class

somf_TCollection (overridden here)

somfPeek Method

Purpose

Determines the object with the “highest” priority in the priority queue, but does not remove it.

IDL Syntax

somf_MOrderableCollectible somfPeek ( );

Description

The **somfPeek** method determines the object with the “highest” priority in the priority queue, but does not remove it from the receiving object.

Parameters

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

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

Return Value

There are two possible valid return values for this method:

somf_MOrderableCollectible
A pointer to the object with the “highest” priority in the priority queue.

SOMF_NIL No object remains in the priority queue.

Example

```
somf_TPriorityQueue pq;
somf_MOrderableCollectible obj;
Environment *ev;

ev = somGetGlobalEnvironment();

pq = somf_TPriorityQueueNew();

/* Add some objects to pq */

/* Look at the highest priority object */
if ((obj = (_somfPeek(pq,ev))) == SOMF_NIL)
    somPrintf(” Nothing is in pq\n”);

_somFree (pq);
_somFree (obj);
```

Original Class

somf_TPriorityQueue

Related Information

Methods: somfPop

somfPop Method

Purpose

Gets the object with the “highest” priority from a given priority queue.

IDL Syntax

somf_MOrderableCollectible somfPop ( );

Description

The **somfPop** method removes the object with the “highest” priority from the specified priority queue, and returns a pointer to it.

Parameters

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

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

Return Value

There are two possible valid return values for this method:

somf_MOrderableCollectible
A pointer to the highest-priority object that was removed from the priority queue.

SOMF_NIL No object remains in the priority queue.

Example

```
somf_TPriorityQueue pq;
somf_MOrderableCollectible obj;
Environment *ev;

ev = somGetGlobalEnvironment();

pq = somf_TPriorityQueueNew();

/* Add some objects to pq */

/* Get the highest priority object */
if ((obj = (_somfPop(pq,ev))) == SOMF_NIL)
    somPrintf(” Nothing is in pq\n”);

_somFree (pq);
_somFree (obj);
```

Original Class

somf_TPriorityQueue

Related Information

Methods: somfPeek, somfReplace

somfRemove Method

Purpose

Removes an object `obj` from a given priority queue.

IDL Syntax

somf_MCollectible somfRemove (in somf_MCollectible *obj*);

Description

The **somfRemove** method removes the specified object `obj` from the priority queue 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_TPriorityQueue_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_TPriorityQueue.**
ev	A pointer to the **Environment** structure for the calling method.
obj	A pointer to the **somf_MCollectible** object to be removed from the priority queue.

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_TPriorityQueue pq;
<your Class which inherits from somf_MOrderableCollectible> obj;
Environment *ev;

ev = somGetGlobalEnvironment();

pq = somf_TPriorityQueueNew();
obj = <your Class that inherits from somf_MOrderableCollectible>New();

/* Add objects to pq */

/* Remove obj from pq */
if (somf_TPriorityQueue_somfRemove(pq,ev,obj) == SOMF_NIL)
    somPrintf(" obj was not in pq\n");

_somFree (pq);
_somFree (obj);
```

Original Class

somf_TCollection (overridden here)

Related Information

Methods: somfRemoveAll

somfRemoveAll Method

Purpose

Removes all of the objects from a given priority queue.

IDL Syntax

void somfRemoveAll ( );

Description

The **somfRemoveAll** method removes all of the objects from the priority queue 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 (example: **somf_TPriorityQueue_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_TPriorityQueue.**

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

Return Value

None.

Example

```
somf_TPriorityQueue pq;
Environment *ev;

ev = somGetGlobalEnvironment();

pq = somf_TPriorityQueueNew();

/* Add objects to pq */

/* Remove all the objects from pq */
_somfRemoveAll(pq,ev);

_somFree (pq);
```

Original Class

somf_TCollection (overridden here)

Related Information

Methods: somfRemove

somfReplace Method

Purpose

Removes the object with the highest priority from a given priority queue, and then inserts an object `obj` into the priority queue.

IDL Syntax

somf_MOrderableCollectible somfReplace (in somf_MOrderableCollectible *obj*);

Description

The **somfReplace** method removes the object with the highest priority from the priority queue represented by the receiving object. It then inserts the given object *obj* into the priority queue.

Parameters

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

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

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

Return Value

There are two possible valid return values for this method:

somf_MOrderableCollectible
A pointer to the object with the “highest” priority that was removed from the priority queue.

SOMF_NIL No object remained in the priority queue when the object *obj* was inserted.

Example

```
somf_TPriorityQueue pq;
<your Class which inherits from somf_MOrderableCollectible> obj;
Environment *ev;

ev = somGetGlobalEnvironment();

pq = somf_TPriorityQueueNew();
obj = <your Class which inherits from somf_MOrderableCollectible>New();

/* Add objects to pq */

if ((_somfReplace(pq,ev,obj)) == SOMF_NIL)
    somPrintf(" pq was empty\n");

_somFree (pq);
_somFree (obj);
```

Original Class

somf_TPriorityQueue

Related Information

Methods: somfPop, somfInsert

somfSetEqualityComparisonFunction Method

Purpose

Sets a method to be called as the equality comparison function when removing objects from the queue, checking whether a given object is a member, and so on.

IDL Syntax

void somfSetEqualityComparisonFunction (in somf_MCollectibleCompareFn *testfn***)**

Description

The **somfSetEqualityComparisonFunction** sets the method that will be called as the equality comparison function when removing objects from the priority queue, checking whether a given object is a member, and so forth. The default method is **somfIsEqual**. Normally, this default method will not need to be changed.

Note: Do not confuse this "equality comparison function" with the **somfCompare** method in **somf_MOrderableCollectible**. This input parameter is *not* used to determine priority.

Parameters

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

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_TPriorityQueue** 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_TPriorityQueue** object.

Return Value

None.

Example

```
somf_TPriorityQueue pq;
Environment *ev;

ev = somGetGlobalEnvironment();

pq = somf_TPriorityQueueNew();

/* Add some objects to pq */

_somfSetEqualityComparisonFunction(pq,ev,
                                   somf_MCollectibleClassData.somfIsEqual);

_somFree (pq);
```

Original Class

somf_TPriorityQueue

Related Information

Methods: somfGetEqualityComparisonFunction

somfTPriorityQueueInitF Method

Purpose

Initializes a new priority queue, given a comparison test method.

IDL Syntax

somf_TPriorityQueue somfTPriorityQueueInitF (
in somf_MOrderableCompareFn *testfn***);**

Description

The **somfTPriorityQueueInitF** method initializes a new priority queue, given a comparison test method that will be used to determine the priority of objects in the priority queue.

Note: This is the only way to set the comparison function used to determine priority for instances of the class. If this method is not used, the **somfIsLessThan** method is used.

Note: You cannot override this method.

Parameters

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

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

testfn The method to be used to determine the "priority" of the objects in the queue. This determines whether "higher priority" objects are removed first or last. Using the **somfIsLessThan** method means that smaller objects are removed first and larger objects are removed last. Using the **somfIsGreaterThan** method reverses this.

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

Return Value

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

Example

```
somf_TPriorityQueue pq1;
Environment *ev;

ev = somGetGlobalEnvironment();

pq1 = somf_TPriorityQueueNew();
_somfTPriorityQueueInitF(pq1,ev,
                somf_MOrderableCollectibleClassData.somfIsLessThan);

_somFree (pq1);
```

Original Class

somf_TPriorityQueue

Related Information

Methods: somfTPriorityQueueInitP

somfTPriorityQueueInitP Method

Purpose

Initializes a new priority queue, setting it equal to another specified priority queue.

IDL Syntax

somf_TPriorityQueue somfTPriorityQueueInitP (in somf_TPriorityQueue *q***);**

Description

The **somfTPriorityQueueInitP** method initializes a new priority queue represented by the receiving object. The method also sets the new priority queue equal to another specified priority queue. This implies that the instance data of the new priority queue will be set equal to those of the source priority queue.

Note: You cannot override this method.

Parameters

receiver	A pointer to an object of class **somf_TPriorityQueue.**
ev	A pointer to the **Environment** structure for the calling method.
q	A pointer to the existing instance of **somf_TPriorityQueue** to which the new priority queue will be set equal.

Return Value

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

Example

```
somf_TPriorityQueue pq1;
somf_TPriorityQueue pq2;
Environment *ev;

ev = somGetGlobalEnvironment();

pq1 = somf_TPriorityQueueNew();
pq2 = somf_TPriorityQueueNew();
_somfTPriorityQueueInitP(pq2,ev,pq1);

_somFree (pq1);
_somFree (pq2);
```

Original Class

somf_TPriorityQueue

Related Information

Methods: somfTPriorityQueueInitF

somf_TPriorityQueueIterator Class

Description

The **somf_TPriorityQueueIterator** class defines an iterator for **somf_TPriorityQueue** that will iterate over all of the objects in a priority queue.

Note: A **somf_TPriorityQueueIterator** iterator does *not* return objects "in order," because a **somf_TPriorityQueue** is only partially ordered in storage.

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

tpqitr

Base

somf_TIterator

Metaclass

SOMClass

Ancestor Classes

somf_TIterator, SOMClass

New Methods

somfTPriorityQueueIteratorInit

Overriding Methods

somfNext
somfFirst
somfRemove

somfFirst Method

Purpose

Resets the iterator and returns the first object in a priority queue.

IDL Syntax

somf_MCollectible somfFirst ( );

Description

The **somfFirst** method resets the **somf_TPriorityQueueIterator** iterator given as the receiving object. The method also returns the first object of the priority queue that corresponds to the specified iterator.

This method resets the iterator to the beginning of the priority queue collection. 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_TPriorityQueueIterator_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_TPriorityQueueIterator.**
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 priority queue collection. Or, SOMF_NIL is returned if the collection is empty.

Example

```
somf_TPriorityQueue pq;
Environment *ev;
somf_TPriorityQueueIterator itr;
somf_MOrderableCollectible itrobj;

ev = somGetGlobalEnvironment();

pq = somf_TPriorityQueueNew();
itr = somf_TPriorityQueueIteratorNew();
_somfTPriorityQueueIteratorInit(itr, ev, pq);

/* Add some object to pq */

/* Iterate through the TPriorityQueue */
itrobj = _somfFirst(itr,ev);
while (itrobj != SOMF_NIL)
{
    /* do something with itrobj */
    itrobj = _somfNext(itr,ev);
}
```

```
_somFree (pq);
_somFree (itr);
```

Original Class

somf_TIterator (overridden here)

Related Information

Methods: somfNext

somfNext Method

Purpose

Gets the next object in a priority queue.

IDL Syntax

somf_MCollectible somfNext ( );

Description

The **somfNext** method determines the next object in the priority queue that corresponds to the iterator represented by the receiving object and, if found, returns a pointer to it. Objects are retrieved in an order that reflects the "ordered–ness" of the priority queue (or the lack of ordering on the priority queue objects).

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 priority queue 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_TPriorityQueueIterator.**
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** in the priority queue.

SOMF_NIL
: The end of the collection has been reached.

Example

```
somf_TPriorityQueue pq;
Environment *ev;
somf_TPriorityQueueIterator itr;
somf_MOrderableCollectible itrobj;

ev = somGetGlobalEnvironment();

pq = somf_TPriorityQueueNew();
itr = somf_TPriorityQueueIteratorNew();
_somfTPriorityQueueIteratorInit(itr, ev, pq);

/* Add some object to pq */
```

```
/* Iterate through the TPriorityQueue */
itrobj = _somfFirst(itr,ev);
while (itrobj != SOMF_NIL)
{
    /* do something with itrobj */

    itrobj = _somfNext(itr,ev);
}

_somFree (pq);
_somFree (itr);
```

Original Class

somf_TIterator (overridden here)

Related Information

Methods: somfFirst

somfRemove Method

Purpose

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

IDL Syntax

void somfRemove ( );

Description

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

The **somfRemove** method is the only way to remove an object from a priority queue 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 priority queue.

If the collection has changed (other than through the use of the **somfRemove** method of this iterator) since the last time **somfFirst** 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_TPriorityQueueIterator_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_TPriorityQueueIterator.**
ev	A pointer to the **Environment** structure for the calling method.

Return Value

None.

Example

```
somf_TPriorityQueue pq;
Environment *ev;
somf_TPriorityQueueIterator itr;

ev = somGetGlobalEnvironment();

pq = somf_TPriorityQueueNew();
itr = somf_TPriorityQueueIteratorNew();
_somfTPriorityQueueIteratorInit(itr, ev, pq);

/* Add some object to pq */

/* Remove the first object in pq */
_somfFirst(itr,ev);
somf_TPriorityQueueIterator_somfRemove(itr,ev);

_somFree (pq);
_somFree (itr);
```

Original Class

somf_TIterator

somfTPriorityQueueIteratorInit Method

Purpose

Initializes a new priority queue iterator, given the priority queue over which it will iterate.

IDL Syntax

somf_TPriorityQueueIterator somfTPriorityQueueIteratorInit (
in somf_TPriorityQueue *h***);**

Description

The **somfTPriorityQueueIteratorInit** method initializes a new **somf_TPriorityQueueIterator** iterator, given the **somf_TPriorityQueue** object over which iteration is needed.

Note: This is one of two ways to initialize a **somf_TPriorityQueueIterator** iterator to point to an instance of a **somf_TPriorityQueue** priority queue collection. The other way is to use the **somf_TPriorityQueue** class's **somfCreateIterator** method described on page 226.

Note: You cannot override this method.

Parameters

receiver	A pointer to an object of class **somf_TPriorityQueueIterator.**
ev	A pointer to the **Environment** structure for the calling method.
h	A pointer to the **somf_TPriorityQueue** object over which the receiving object will iterate.

Return Value

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

Example

```
somf_TPriorityQueue pq;
Environment *ev;
somf_TPriorityQueueIterator itr;

ev = somGetGlobalEnvironment();

pq = somf_TPriorityQueueNew();
itr = somf_TPriorityQueueIteratorNew();
_somfTPriorityQueueIteratorInit(itr, ev, pq);

_somFree (pq);
_somFree (itr);
```

Original Class

somf_TPriorityQueueIterator