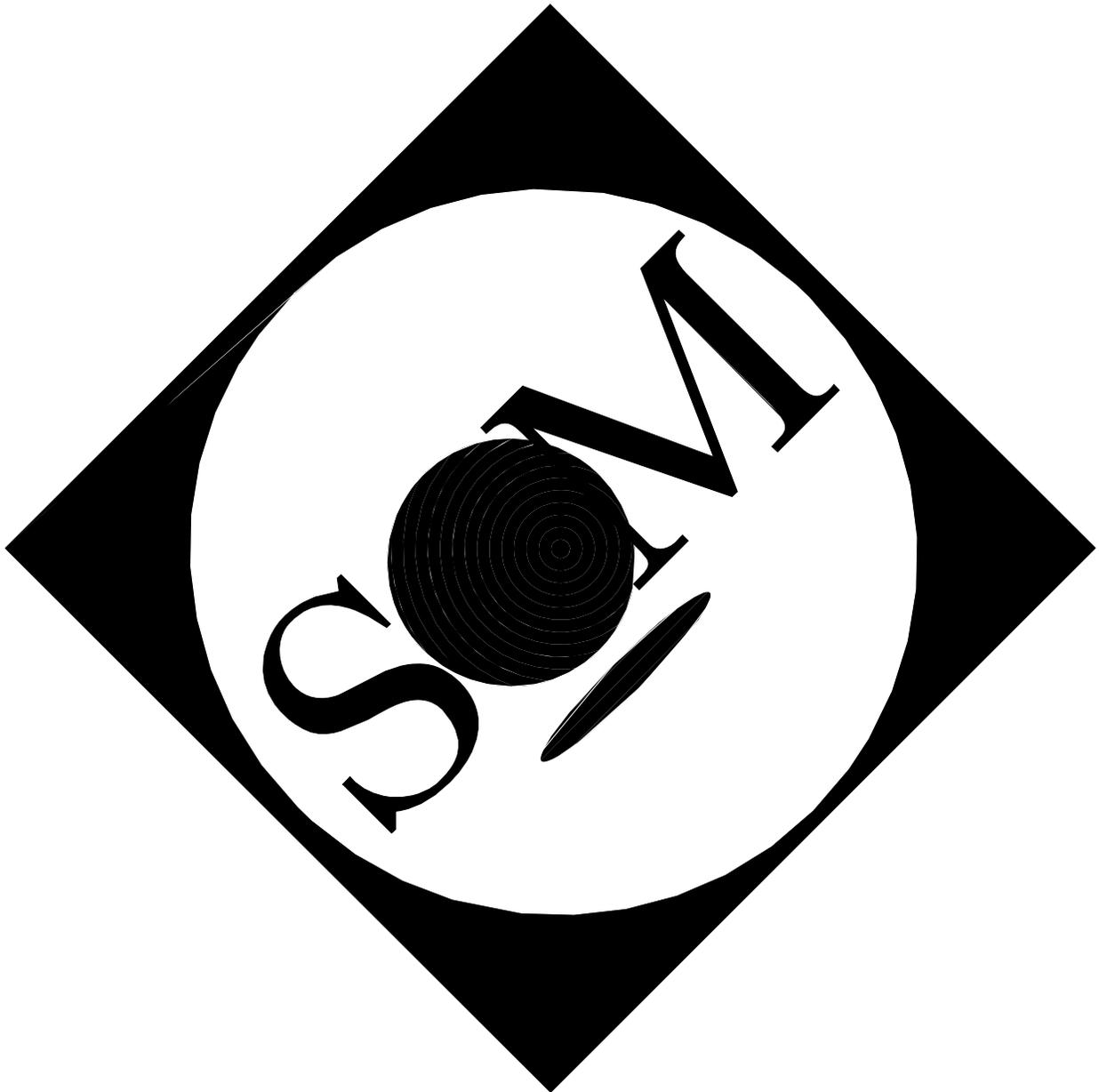


Programmer's Reference, Volume IV: SOM Collection Classes

SOMobjects Version 3.0



Note: Before using this information and the product it supports, be sure to read the general information under “Notices” on page iii.

First Edition (March 1996)

This edition of *Programmer's Reference, Volume IV: SOM Collection Classes* applies to SOMObjects Developer Toolkit for SOM Version 3.0 and to all subsequent releases of the product until otherwise indicated in new releases or technical newsletters.

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law: IBM CORPORATION PROVIDES THIS MANUAL “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

IBM Corporation does not warrant that the contents of this publication or the accompanying source code examples, whether individually or as one or more groups, will meet your requirements nor that the publication or the accompanying source code examples are error-free.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes are incorporated in new editions of the publication. IBM Corporation might make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

This publication might contain references to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM Corporation intends to announce such IBM products, programming, or services in your country. Any reference to an IBM licensed program in this publication is not intended to state or imply that you can use only the IBM licensed program. You can use any functionally equivalent program instead.

To initiate changes to this publication, post the CompuServe™ IBMSOM forum or send email to sombug@austin.ibm.com. Otherwise, address comments to IBM Corporation, Internal Zip 1002, 11400 Burnet Road, Austin, Texas 78758-3493. IBM Corporation may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Requests for copies of this publication and for technical information about IBM products should be made to your IBM Authorized Dealer or your IBM Marketing representative.

© Copyright IBM Corporation 1995. All rights reserved.

Notice to U.S. Government Users — Documentation Related to Restricted Rights — Use, duplication, or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corporation.

Notices

IBM Corporation may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Commercial Relations, IBM Corporation, Purchase, NY 10577.

COPYRIGHT LICENSE: This publication contains printed sample application programs in source language, which illustrate AIX, OS/2, or Windows programming techniques. You may copy and distribute these sample programs in any form without payment to IBM Corporation, for the purposes of developing, using, marketing, or distributing application programs conforming to the AIX, OS/2, or Windows application programming interface.

Each copy of any portion of these sample programs or any derivative work, which is distributed to others, must include a copyright notice as follows: "© (your company name) (current year), All Rights Reserved." However, the following copyright notice protects this documentation under the Copyright Laws of the United States and other countries which prohibit such actions as, but not limited to, copying, distributing, modifying, and making derivative works.

References in this publication to IBM products, program, or services do not imply that IBM Corporation intends to make these available in all countries in which it operates.

Any reference to IBM licensed programs, products, or services is not intended to state or imply that only IBM licensed programs, products, or services can be used. Any functionally-equivalent product, program or service that does not infringe upon any of the IBM Corporation intellectual property rights may be used instead of the IBM Corporation product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM Corporation, are the user's responsibility.

IBM Corporation may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries in writing to the:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, New York 10594, USA

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Department 931S
11400 Burnet Road
Austin, Texas 78758 USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Asia-Pacific users can inquire, in writing, to the:

IBM Director of Intellectual Property and Licensing
IBM World Trade Asia Corporation,
2-31 Roppongi 3-chome,
Minato-ku, Tokyo 106, Japan

This publication contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Trademarks and Acknowledgements

AIX is a trademark of International Business Machines Corporation.

CompuServe is a trademark of CompuServe, Inc.

FrameViewer is a trademark of Frame Technology.

IBM is a registered trademark of International Business Machines Corporation.

OS/2 is a trademark of International Business Machines Corporation.

SOM is a trademark of International Business Machines Corporation.

SOMobject is a trademark of International Business Machines Corporation.

Windows is a trademark of Microsoft Corporation.

Table of Contents

Notices	iii
Trademarks and Acknowledgements	iv
somf_MCollectible Class	1
somfClone Method	2
somfClonePointer Method	3
somfHash Method	4
somfIsEqual Method	5
somfIsNotEqual Method	6
somfIsSame Method	7
somf_MLinkable Class	8
somfGetNext Method	9
somfGetPrevious Method	10
somfMLinkableInit Method	11
somfSetNext Method	12
somfSetPrevious Method	13
somf_MOrderableCollectible Class	14
somfCompare Method	15
somfIsGreaterThan Method	17
somfIsGreaterThanOrEqualTo Method	18
somfIsLessThan Method	19
somfIsLessThanOrEqualTo Method	20
somf_TAssoc Class	21
somfGetKey Method	22
somfGetValue Method	23
somfSetKey Method	24
somfSetValue Method	25
somfTAssocInitM Method	26
somfTAssocInitMM Method	27
somf_TCollectibleLong Class	28
somfGetValue Method	29
somfHash Method	30
somfIsEqual Method	31
somfSetValue Method	32
somfTCollectibleLongInit Method	33
somf_TCollection Class	34
somfAdd Method	35
somfAddAll Method	36
somfCount Method	37
somfCreateIterator Method	38
somfDeleteAll Method	39
somfIsEqual Method	40
somfMember Method	41
somfRemove Method	42
somfRemoveAll Method	43
somfSetTestFunction Method	44
somfTCollectionInit Method	45
somfTestFunction Method	46
somf_TDeque Class	47
somfAdd Method	49
somfAddAfter Method	50

somfAddBefore Method	51
somfAddFirst Method	52
somfAddLast Method	53
somfAfter Method	54
somfAssign Method	55
somfBefore Method	56
somfCount Method	57
somfCreateliterator Method	58
somfCreateNewLink Method	59
somfCreateSequenceliterator Method	60
somfDeleteAll Method	61
somfFirst Method	62
somfInsert Method	63
somfLast Method	64
somfMember Method	65
somfPop Method	66
somfPush Method	67
somfRemove Method	68
somfRemoveAll Method	69
somfRemoveFirst Method	70
somfRemoveLast Method	71
somfRemoveQ Method	72
somfTDequeInitD Method	73
somfTDequeInitF Method	74
somf_TDequeIterator Class	75
somfFirst Method	76
somfLast Method	77
somfNext Method	78
somfPrevious Method	80
somfRemove Method	81
somfTDequeIteratorInit Method	82
somf_TDequeLinkable Class	83
somfGetValue Method	84
somfSetValue Method	85
somfTDequeLinkableInitDD Method	86
somfTDequeLinkableInitDDM Method	87
somf_TDictionary Class	89
somfAdd Method	91
somfAddKeyValuePairMM Method	92
somfAddKeyValuePairMMB Method	94
somfAssign Method	96
somfCopyImplementation Method	97
somfCount Method	98
somfCreateliterator Method	99
somfCreateNewImplementationF Method	100
somfCreateNewImplementationFL Method	102
somfCreateNewImplementationFLL Method	104
somfCreateNewImplementationFLLL Method	106
somfDeleteAll Method	108
somfDeleteAllKeys Method	109
somfDeleteAllValues Method	110
somfDeleteKey Method	111
somfGetHashFunction Method	113
somfKeyAtM Method	114

somfKeyAtMF Method	115
somfMember Method	117
somfRemove Method	118
somfRemoveAll Method	119
somfSetHashFunction Method	120
somfTDictionaryInitD Method	121
somfTDictionaryInitF Method	122
somfTDictionaryInitFL Method	123
somfTDictionaryInitFLL Method	125
somfTDictionaryInitL Method.	127
somfTDictionaryInitLL Method.	128
somfTDictionaryInitLLF Method	129
somfValueAt Method	131
somf_TDictionaryIterator Class	132
somfFirst Method.	133
somfNext Method	135
somfRemove Method	137
somfTDictionaryIteratorInit Method	138
somf_THashTable Class	139
somfAddMM Method	141
somfAddMMB Method.	142
somfAssign Method.	144
somfCount Method	145
somfDelete Method	146
somfDeleteAll Method	147
somfDeleteAllKeys Method	148
somfDeleteAllValues Method	149
somfGetGrowthRate Method.	150
somfGetHashFunction Method	151
somfGetRehashThreshold Method	152
somfGrow Method	153
somfMember Method	154
somfRemove Method	155
somfRemoveAll Method	156
somfRetrieve Method	157
somfSetGrowthRate Method.	158
somfSetHashFunction Method	159
somfSetRehashThreshold Method	160
somfTHashTableInitFL Method.	161
somfTHashTableInitFLL Method.	162
somfTHashTableInitFLLL Method.	164
somfTHashTableInitH Method.	166
somf_THashTableIterator Class	167
somfFirst Method.	168
somfNext Method	170
somfRemove Method	172
somfTHashTableIteratorInit Method	173
somf_TIterator Class	174
somfFirst Method.	175
somfNext Method	176
somfRemove Method	177
somf_TPrimitiveLinkedList Class	178
somfAddAfter Method	179
somfAddBefore Method	180

somfAddFirst Method	181
somfAddLast Method	182
somfAfter Method	183
somfBefore Method	184
somfCount Method	185
somfFirst Method	186
somfLast Method	187
somfRemove Method	188
somfRemoveAll Method	189
somfRemoveFirst Method	190
somfRemoveLast Method	191
somf_TPrimitiveLinkedListIterator Class	192
somfFirst Method	193
somfLast Method	194
somfNext Method	195
somfPrevious Method	196
somfTPrimitiveLinkedListIteratorInit Method	197
somf_TPriorityQueue Class	199
somfAdd Method	201
somfAssign Method	202
somfCount Method	203
somfCreateIterator Method	204
somfDeleteAll Method	205
somfGetEqualityComparisonFunction Method	206
somfInsert Method	207
somfMember Method	208
somfPeek Method	209
somfPop Method	210
somfRemove Method	211
somfRemoveAll Method	212
somfReplace Method	213
somfSetEqualityComparisonFunction Method	214
somfTPriorityQueueInitF Method	215
somfTPriorityQueueInitP Method	216
somf_TPriorityQueueIterator Class	217
somfFirst Method	218
somfNext Method	219
somfRemove Method	220
somfTPriorityQueueIteratorInit Method	221
somf_TSequence Class	223
somfAdd Method	224
somfAfter Method	225
somfBefore Method	226
somfCount Method	227
somfCreateIterator Method	228
somfDeleteAll Method	229
somfFirst Method	230
somfLast Method	231
somfOccurrencesOf Method	232
somfRemove Method	233
somfRemoveAll Method	234
somfTSequenceInit Method	235
somf_TSequenceIterator Class	236
somfFirst Method	237

somfLast Method	238
somfNext Method	239
somfPrevious Method	240
somfRemove Method	241
somf_TSet Class	243
somfAdd Method	245
somfAssign Method	246
somfCount Method	247
somfCreateIterator Method	248
somfDeleteAll Method	249
somfDifferenceS Method	250
somfDifferenceSS Method	251
somfGetHashFunction Method	252
somfIntersectionS Method	253
somfIntersectionSS Method	254
somfMember Method	255
somfRehash Method	256
somfRemove Method	257
somfRemoveAll Method	258
somfSetHashFunction Method	259
somfTSetInitF Method	260
somfTSetInitFL Method	261
somfTSetInitL Method	262
somfTSetInitLF Method	263
somfTSetInitS Method	264
somfUnionS Method	265
somfUnionSS Method	266
somfXorS Method	267
somfXorSS Method	268
somf_TSetIterator Class	269
somfFirst Method	270
somfNext Method	271
somfRemove Method	273
somfTSetIteratorInit Method	274
somf_TSortedSequence Class	275
somfAdd Method	277
somfAfter Method	278
somfAssign Method	279
somfBefore Method	280
somfCount Method	281
somfCreateIterator Method	282
somfCreateSequenceIterator Method	283
somfCreateSortedSequenceNode Method	284
somfDeleteAll Method	285
somfFirst Method	286
somfGetSequencingFunction Method	287
somfLast Method	288
somfMember Method	289
somfOccurrencesOf Method	290
somfRemove Method	291
somfRemoveAll Method	292
somfSetSequencingFunction Method	293
somfTSortedSequenceInitF Method	294
somfTSortedSequenceInitS Method	295

somf_TSortedSequenceIterator Class	296
somfFirst Method	297
somfLast Method	299
somfNext Method	300
somfPrevious Method	302
somfRemove Method	303
somfStartHere Method	304
somfTSortedSequenceIteratorInit Method	305
somf_TSortedSequenceNode Class	307
somfGetKey Method	308
somfGetLeftChild Method	309
somfGetParent Method	310
somfGetRed Method	311
somfGetRightChild Method	312
somfSetKey Method	313
somfSetLeftChild Method	314
somfSetParent Method	315
somfSetRed Method	316
somfSetRedOn Method	317
somfSetRightChild Method	318
somfTSortedSequenceNodeInitT Method	319
somfTSortedSequenceNodeInitTM Method	320
somfTSortedSequenceNodeInitTMT Method	321
Index	323

somf_MCollectible Class

The **somf_MCollectible** class represents the generic class from which most other collection classes are derived. It can be critical for subclasses to define some or all of the methods presented below.

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

The reason new classes inherit from **somf_MCollectible** is that instances of the inheriting class can be inserted into a main collection classes. All classes that inherit from **somf_MCollectible** must override either **somflsEqual Method** or **somflsSame Method**, depending on the method the new class plans to use for comparison. The **somfHash Method** will probably need to be overridden. This class is not thread-safe.

File Stem

mcollect

Base

SOMObject Class

Metaclass

SOMClass Class

Ancestor Classes

SOMObject Class

New Methods

somfClone Method
somfClonePointer Method
somfHash Method
somflsEqual Method
somflsSame Method
somflsNotEqual Method

Typedefs

The following typedefs are defined in the **somf_MCollectible** class:

somf_MCollectibleCompareFn

A method pointer to a **somflsEqual** or **somflsSame** method.

somf_MCollectibleHashFn

A method pointer to a **somfHash** method.

Defines

The following defines originate in this class:

SOMF_NIL

A representation of nil used by the collection classes.

SOMF_CALL_COMPARE_FN

A define to help call the method pointed to by **somf_MCollectibleCompareFn**.

SOMF_CALL_HASH_FN

A define to help call the method pointed to by **somf_MCollectibleHashFn**.

somfClone Method

Provides a general polymorphic duplication operation.

IDL Syntax

```
somf_MCollectible somfClone ();
```

Description

The **somfClone** method provides a general polymorphic duplication operation.

Parameters

receiver

A pointer to an object of class **somf_MCollectible**.

ev

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

Return Value

This method returns a pointer to a new object of the same class as the receiver. The receiving object must be an object of the **somf_MCollectible** class or of a class that inherits from **somf_MCollectible**. The **somfClone** method determines the true class of the receiver and creates a new instance of that class, and then returns a pointer to that instance.

Example

```
somf_MCollectible clone;
somf_TSortedSequence ss;
Environment *ev;

ev = somGetGlobalEnvironment();

ss = somf_TSortedSequenceNew();

clone = _somfClone(ss, ev);
somPrintf("\n Clone returned Class %s\n",
         _somGetClassName(clone));

_somFree (ss);
_somFree (clone);
```

Original Class

somf_MCollectible Class

Related Information

somfClonePointer Method

somfClonePointer Method

Returns a pointer to a Clone.

IDL Syntax

```
somf_MCollectible somfClonePointer (in somf_MCollectible clonee);
```

Description

The **somfClonePointer** method returns a pointer to a Clone.

Note: You cannot override this method.

Parameters

receiver

A pointer to an object of class **somf_MCollectible**.

ev

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

clonee

A pointer to the **somf_MCollectible** to be cloned.

Return Value

There are two possible valid return values for this method:

- **somf_MCollectible**, a pointer to a new instance of the calling class, which inherits from the **somf_MCollectible** class.
- **SOMF_NIL**, the *clonee* is nil, so a clone could not be created.

Example

```
somf_MCollectible clone;
somf_TSortedSequence ss;
Environment *ev;

ev = somGetGlobalEnvironment();
ss = somf_TSortedSequenceNew();

clone = _somfClonePointer(ss, ev, ss);
somPrintf("\n Clone returned Class %s\n",
         _somGetClassName(clone));

_somFree (ss);
_somFree (clone);
```

Original Class

somf_MCollectible Class

Related Information

somfClone Method

somfHash Method

\Returns a value suitable for use as a hashing probe for the receiving object.

IDL Syntax

```
long somfHash ( );
```

Description

The **somfHash** method returns a value suitable for use as a hashing probe for the receiving object.

This method should be overridden if a class inherits from **somf_MCollectible**. The default function will simply return the address of the object. The default function is almost certainly not adequate if you are overriding **somfIsEqual Method**, because you need to make sure that all objects that are equal to each other return the same hash value.

Parameters

receiver

A pointer to an object of class **somf_MCollectible**.

ev

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

Return Value

This method returns the hash value for the receiving object.

Example

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

ev = somGetGlobalEnvironment();
obj = <Your class which inherits from somf_MCollectible>New();

somPrintf(" The Hashing probe for obj is %d\n", _somfHash(obj,ev));

_somFree (obj);
```

Original Class

somf_MCollectible Class

somflsEqual Method

\Returns TRUE if a given *obj* is isomorphic to the receiving object.

IDL Syntax

```
boolean somflsEqual (in somf_MCollectible obj);
```

Description

The **somflsEqual** method returns TRUE if another specified *obj* is isomorphic to the receiving object. Most utility classes allow you to specify what methods to use when comparing objects for insertion, deletion. The choice is to use either the **somflsEqual** method, or the **somflsSame Method**.

This method must be overridden if a class inherits from **somf_MCollectible**. If it is not overridden, and this method is used, an error message is written and processing will end.

Parameters

receiver

A pointer to an object of class **somf_MCollectible**.

ev

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

obj

A pointer to the **somf_MCollectible** object that the receiving object will be compared against.

Return Value

This method returns a boolean value:

- TRUE, *obj* is equal to the receiving object.
- FALSE, *obj* is not equal to the receiving object.

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. The following example shows how to use this method once it is overridden.

```
<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();

if (_somflsEqual(obj, ev, obj2))
    somPrintf(" obj is equal to obj2\n");

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

Original Class

somf_MCollectible Class

Related Information

somflsNotEqual Method

somflsNotEqual Method

Returns TRUE if a specified *obj* is not isomorphic to the receiving object.

IDL Syntax

```
boolean somflsNotEqual (in somf_MCollectible obj);
```

Description

The **somflsNotEqual** method returns TRUE if the specified object *obj* is not isomorphic to the receiving object. This method uses the **somflsEqual**. If a class inherits from **somf_MCollectible**, **somflsEqual** must be overridden for this method to work.

Parameters

receiver

A pointer to an object of class **somf_MCollectible**.

ev

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

obj

A pointer to the **somf_MCollectible** object that the receiving object will be compared against.

Return Value

This method returns a boolean value:

- TRUE, *obj* is not equal to the receiving object.
- FALSE, *obj* is equal to the receiving object.

Example

```
<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();

if (_somflsNotEqual(obj, ev, obj2))
    somPrintf(" obj is NOT equal to obj2\n");

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

Original Class

somf_MCollectible Class

Related Information

somflsEqual Method

somflsSame Method

Performs a pointer comparison between the receiving object and another specified object, *obj*.

IDL Syntax

```
boolean somflsSame (in somf_MCollectible obj);
```

Description

The **somflsSame** method performs a pointer comparison between the receiving object and another specified *obj*.

Parameters

receiver

A pointer to an object of class **somf_MCollectible**.

ev

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

obj

A pointer to the **somf_MCollectible** object that the receiving object will be compared against.

Return Value

This method returns a boolean value:

- TRUE, *obj* is the same as the receiving object.
- FALSE, *obj* is not the same as the receiving object.

Example

```
<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();

if (_somflsSame(obj, ev, obj2))
    somPrintf(" obj is the same as obj2\n");

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

Original Class

somf_MCollectible Class

somf_MLinkable Class

This class defines the general characteristics of objects that contain links. For example, **somf_TPrimitiveLinkedListIterator Class** uses **somf_MLinkable**.

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

Other classes would inherit from **somf_MLinkable** if the user plans to link one class to another class, either in a **somf_TPrimitiveLinkedList** or through another class.

This class is not thread-safe.

File Stem

mlink

Base

SOMObject Class

Metaclass

SOMClass Class

Ancestor Classes

SOMObject Class

New Methods

somfGetNext Method
somfGetPrevious Method
somfMLinkableInit Method
somfSetNext Method
somfSetPrevious Method

Overriding Methods

somDefaultInit Method

somfGetNext Method

Gets a pointer to the next **somf_MLinkable** object.

IDL Syntax

```
somf_MLinkable somfGetNext ( );
```

Description

The **somfGetNext** method gets a pointer to the next object of class **somf_MLinkable**.

Parameters

receiver

A pointer to an object of class **somf_MLinkable**.

ev

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

Return Value

This method returns the pointer to the next **somf_MLinkable** object.

Example

```
somf_MLinkable ml;
somf_MLinkable ml2;
Environment *ev;

ev = somGetGlobalEnvironment();

ml = somf_MLinkableNew();

/* Determine ml's next pointer */
ml2 = _somfGetNext(ml, ev);

_somFree (ml);
```

Original Class

somf_MLinkable Class

Related Information

somfSetNext Method

somfSetPrevious Method

somfGetPrevious Method

Gets a pointer to the previous **somf_MLinkable** object.

IDL Syntax

```
somf_MLinkable somfGetPrevious ( );
```

Description

The **somfGetPrevious** method returns a pointer to the previous object of class **somf_MLinkable**.

Parameters

receiver

A pointer to an object of class **somf_MLinkable**.

ev

A pointer to the Environment structure for the calling method.

Return Value

This method returns the pointer the previous **somf_MLinkable** object.

Example

```
somf_MLinkable ml;
somf_MLinkable ml2;
Environment *ev;

ev = somGetGlobalEnvironment();

ml = somf_MLinkableNew();

/* Determine ml's previous pointer */
ml2 = _somfGetPrevious(ml, ev);

_somFree (ml);
```

Original Class

somf_MLinkable Class

Related Information

somfGetNext Method

somfSetPrevious Method

sopfMLinkableInit Method

Initializes a new **sopf_MLinkable** object, given pointers to its next and previous objects.

IDL Syntax

```
sopf_MLinkable sopfMLinkableInit (
    in sopf_MLinkable n,
    in sopf_MLinkable p);
```

Description

The **sopfMLinkableInit** method initializes a new object of class **sopf_MLinkable**, given pointers to the new object's next and previous **sopf_MLinkable** objects.

Note: You cannot override this method.

Parameters

receiver

A pointer to an object of class **sopf_MLinkable**.

ev

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

n

A pointer to the next **sopf_MLinkable** object.

p

A pointer to the previous **sopf_MLinkable** object.

Return Value

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

Example

```
sopf_MLinkable ml;
Environment *ev;

ev = somGetGlobalEnvironment();

ml = sopf_MLinkableNew();
_sopfMLinkableInit(ml, ev, SOMF_NIL, SOMF_NIL);

_sopfFree (ml);
```

Original Class

sopf_MLinkable Class

somfSetNext Method

Sets a link pointer to the next **somf_MLinkable** object, given a pointer to the object that should come after the receiving object.

IDL Syntax

```
void somfSetNext (in somf_MLinkable aLink);
```

Description

The **somfSetNext** method sets a link pointer to the next object of class **somf_MLinkable**, given a pointer to the object that should follow the receiving object.

Parameters

receiver

A pointer to an object of class **somf_MLinkable**.

ev

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

aLink

A pointer to the **somf_MLinkable** object which should be next after the receiving object.

Example

```
somf_MLinkable m1;
somf_MLinkable m2;
Environment *ev;

ev = somGetGlobalEnvironment();

m1 = somf_MLinkableNew();
m2 = somf_MLinkableNew();

/* Set m1's next and previous pointers */

/* Set m2 to point to m1 as the next link */
_somfSetNext(m2, ev, m1);

_somFree (m1);
_somFree (m2);
```

Original Class

somf_MLinkable Class

Related Information

somfGetNext Method

somfSetPrevious Method

somfSetPrevious Method

Sets a link pointer to the previous **somf_MLinkable** object, given a pointer to the object that should come before the receiving object.

IDL Syntax

```
void somfSetPrevious (in somf_MLinkable aLink);
```

Description

The **somfSetPrevious** method sets a link pointer to the previous object of class **somf_MLinkable**, given a pointer to the object that should precede the receiving object.

Parameters

receiver

A pointer to an object of class **somf_MLinkable**.

ev

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

aLink

A pointer to the **somf_MLinkable** object, which should be previous to the receiving object.

Example

```
somf_MLinkable m1;
somf_MLinkable m2;
Environment *ev;

ev = somGetGlobalEnvironment();

m1 = somf_MLinkableNew();
m2 = somf_MLinkableNew();

/* Set m1's next and previous pointers */

/* Set m2 to point to m1 as the previous link */
_somfSetPrevious(m2, ev, m1);

_somFree (m1);
_somFree (m2);
```

Original Class

somf_MLinkable Class

Related Information

somfGetPrevious Method

somfSetNext Method

somf_MOrderableCollectible Class

Characteristics of the **somf_MOrderableCollectible** class should be mixed into objects that might need to be ordered. Objects passed to an instance of **somf_TPriorityQueue Class** or **somf_TSortedSequence Class** must have **somf_MOrderableCollectible** mixed into them.

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

A class will inherit from **somf_MOrderableCollectible** if it represents an element in an ordered collection. All classes that inherit from **somf_MOrderableCollectible** must override the **somflsEqual** method that is inherited from **somf_MCollectible Class**, and **somflsLessThan** and **somflsGreaterThan** methods. This class is not thread-safe.

File Stem

morder

Base

somf_MCollectible Class

Metaclass

SOMClass Class

Ancestor Classes

somf_MCollectible Class
SOMObject Class

New Methods

somflsGreaterThan Method
somflsLessThan Method
somfCompare Method
somflsGreaterThanOrEqualTo Method
somflsGreaterThanOrEqualTo Method

Typedefs

The following typedefs are defined in the **somf_MOrderableCollectible** class:

- **somf_MOrderableCompareFn**, a method pointer to a **somflsLessThan** or **somflsGreaterThan** method.
- **somf_MBetterOrderableCompareFn**, a method pointer to a **somfCompare** method.

Enums

The following enum is defined in this class:

- **EComparisonResult**, an enum with the values: **kLessThan**, **kEqual**, **kGreaterThan**

Defines

The following defines originate in this class:

SOMF_CALL_ORDERABLE_COMPARE_FN

A define to help call the method pointed to by **somf_MOrderableCompareFn**.

SOMF_CALL_BETTER_ORDERABLE_COMPARE_FN

A define to help call the method pointed to by **somf_MBetterOrderableCompareFn**.

somfCompare Method

Compares a specified *obj* to the receiving object, and returns a value indicating *obj*'s comparative size.

IDL Syntax

```
EComparisonResult somfCompare (in somf_MOrderableCollectible obj);
```

Description

The **somfCompare** method compares the specified *obj* to the receiving object. The return value indicates whether *obj* is greater than, less than, or equal to the receiving object.

The **somfIsEqual** method inherited from **somf_MCollectible Class**, as well as methods **somfIsLessThan** and **somfIsGreaterThan** of the **somf_MOrderableCollectible** class, must be overridden before this method will work.

Parameters

receiver

A pointer to an object of class **somf_MOrderableCollectible**.

ev

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

obj

A pointer to the object to which the receiving object will be compared.

Return Value

There are three possible valid return values for this method:

- **kLessThan**, *obj* is less than the receiving object.
- **kEqual**, *obj* is equal to the receiving object.
- **kGreaterThan**, *obj* is greater than the receiving object.

Example

```
<Your Class which inherits from somf_MOrderableCollectible> a1;
<Your Class which inherits from somf_MOrderableCollectible> a2;
Environment *ev;
ev = somGetGlobalEnvironment();
a1 = <Your Class which inherits from
        somf_MOrderableCollectible>New();
a2 = <Your Class which inherits from
        somf_MOrderableCollectible>New();
/* Set a1 and a2 as you wish */
/* Compare a1 and a2 */
if ( _somfCompare(a2, ev, a1) == somf_MOrderableCollectible_kLessThan)
    somPrintf(" a1 is less than a2\n");
else
    somPrintf(" a1 is NOT less than a2\n");
if ( _somfCompare(a1, ev, a2) ==
    somf_MOrderableCollectible_kGreaterThan)
    somPrintf(" a2 is greater than a1\n");
else
    somPrintf(" a2 is NOT greater than a1");
if ( _somfCompare(a2, ev, a2) == somf_MOrderableCollectible_kEqual)
    somPrintf(" a2 is equal a2\n");
else
    somPrintf(" a2 is NOT equal to a2");
_somFree (a1);
_somFree (a2);
```

somfCompare Method

Original Class

somf_MOrderableCollectible Class

Related Information

somfIsEqual Method

somfIsGreaterThan Method

somfIsLessThan Method

somflsGreaterThan Method

Compares two objects and returns TRUE if a given *obj* is “greater than” the receiving object.

IDL Syntax

```
boolean somflsGreaterThan (in somf_MOrderableCollectible obj);
```

Description

The **somflsGreaterThan** method returns TRUE if the specified object is “greater than” the receiving object. This method must be overridden if a class inherits from **somf_MOrderableCollectible**. If not, an error message is written and processing will end.

Parameters

receiver

A pointer to an object of class **somf_MOrderableCollectible**.

ev

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

obj

A pointer to the object to which the receiving object will be compared.

Return Value

This method returns the boolean values TRUE or FALSE, depending on if *obj* “Is Greater Than” the receiving object.

Example

You cannot use this method directly from this class; it must be overridden. The following example shows how you would use this method once it is overridden.

```
<Your Class which inherits from somf_MOrderableCollectible> a1;
<Your Class which inherits from somf_MOrderableCollectible> a2;
Environment *ev;

ev = somGetGlobalEnvironment();
a1 = <Your Class which inherits from
                                somf_MOrderableCollectible>New();
a2 = <Your Class which inherits from
                                somf_MOrderableCollectible>New();

/* Set a1 and a2 as you wish */
/* Compare a1 and a2 */
if ( _somflsGreaterThan(a2, ev, a1)
    somPrintf(" a1 is greater than a2\n");
else
    somPrintf(" a1 is NOT greater than a2\n");

_somFree (a1);
_somFree (a2);
```

Original Class

somf_MOrderableCollectible Class

Related Information

somflsEqual Method

somflsGreaterThanOrEqualTo Method

somflsLessThan Method

somflsGreaterThanOrEqualTo Method

Compares two objects and returns TRUE if a specified *obj* is “greater than” or “equal to” the receiving object.

IDL Syntax

```
boolean somflsGreaterThanOrEqualTo (in somf_MOrderableCollectible obj);
```

Description

The **somflsGreaterThanOrEqualTo** method returns TRUE if a specified object *obj* is “greater than” or “equal to” the receiving object.

Parameters

receiver

A pointer to an object of class **somf_MOrderableCollectible**.

ev

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

obj

A pointer to the object to which the receiving object will be compared.

Return Value

This method returns the Boolean values TRUE or FALSE, depending on whether *obj* “Is Greater Than” or “Is Equal To” the receiving object.

Example

```
<Your Class which inherits from somf_MOrderableCollectible> a1;
<Your Class which inherits from somf_MOrderableCollectible> a2;
Environment *ev;

ev = somGetGlobalEnvironment();

a1 = <Your Class which inherits from
      somf_MOrderableCollectible>New();
a2 = <Your Class which inherits from
      somf_MOrderableCollectible>New();

/* Set a1 and a2 as you wish */

/* Compare a1 and a2 */
if ( _somflsGreaterThanOrEqualTo(a2, ev, a1)
    somPrintf(" a1 is greater than or equal to a2\n");
else
    somPrintf(" a1 is NOT greater than or equal to a2\n");

_somFree (a1);
_somFree (a2);
```

Original Class

somf_MOrderableCollectible Class

Related Information

somflsEqual Method
somflsGreaterThanOrEqualTo Method
somflsLessThanOrEqualTo Method

somflsLessThan Method

Compares two objects and returns TRUE if a given *obj* is “less than” the receiving object.

IDL Syntax

```
boolean somflsLessThan (in somf_MOrderableCollectible obj);
```

Description

The **somflsLessThan** method returns TRUE if the specified object is less than the receiving object. This method must be overridden if a class inherits from **somf_MOrderableCollectible**. If not, an error message is written and processing will end.

Parameters

receiver

A pointer to an object of class **somf_MOrderableCollectible**.

ev

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

obj

A pointer to the object to which the receiving object will be compared.

Return Value

This method returns the boolean values TRUE or FALSE, depending on whether *obj* “Is Less Than” the receiving object.

Example

You cannot use this method directly from this class; it must be overridden. The following example shows how you would use this method once it is overridden.

```
<Your Class which inherits from somf_MOrderableCollectible> a1;
<Your Class which inherits from somf_MOrderableCollectible> a2;
Environment *ev;

ev = somGetGlobalEnvironment();

a1 = <Your Class which inherits from
      somf_MOrderableCollectible>New();
a2 = <Your Class which inherits from
      somf_MOrderableCollectible>New();

/* Set a1 and a2 as you wish */

/* Compare a1 and a2 */
if ( _somflsLessThan(a2, ev, a1))
    somPrintf(" a1 is less than a2\n");
else
    somPrintf(" a1 is NOT less than a2\n");

_somFree (a1);
_somFree (a2);
```

Original Class

somf_MOrderableCollectible Class

Related Information

somflsEqual Method

somflsGreaterThan Method

somflsLessThanOrEqualTo Method

somflsLessThanOrEqualTo Method

Compares two objects and returns TRUE if a given *obj* is “less than” or “equal to” the receiving object.

IDL Syntax

```
boolean somflsLessThanOrEqualTo (in somf_MOrderableCollectible obj);
```

Description

The **somflsLessThanOrEqualTo** method returns TRUE if a specified object *obj* is “less than” or “equal to” the receiving object.

Parameters

receiver

A pointer to an object of class **somf_MOrderableCollectible**.

ev

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

obj

A pointer to the object to which the receiving object will be compared.

Return Value

This method returns the boolean values TRUE or FALSE, depending on whether the specified *obj* “Is Less Than” or “Is Equal To” the receiving object.

Example

```
<Your Class which inherits from somf_MOrderableCollectible> a1;
<Your Class which inherits from somf_MOrderableCollectible> a2;
Environment *ev;

ev = somGetGlobalEnvironment();

a1 = <Your Class which inherits from
      somf_MOrderableCollectible>New();
a2 = <Your Class which inherits from
      somf_MOrderableCollectible>New();

/* Set a1 and a2 as you wish */

/* Compare a1 and a2 */
if ( _somflsLessThanOrEqualTo(a2, ev, a1)
    somPrintf(" a1 is less than or equal to a2\n");
else
    somPrintf(" a1 is NOT less than or equal to a2\n");

_somFree (a1);
_somFree (a2);
```

Original Class

somf_MOrderableCollectible Class

Related Information

somflsLessThan Method
somflsGreaterThan Method
somflsEqual Method

sopf_TAssoc Class

An object of class **sopf_TAssoc** is used to hold a (*key*, *value*) pair of objects. Typically, these structures are owned by some other higher-level object; specifically, the **sopf_THashTable Class** and **sopf_TDictionary Class** use pairs that are actually objects of the **sopf_TAssoc** class.

Objects of the **sopf_TAssoc** class are usually not returned to the user. However, users implementing their own classes to hold pairs of objects might wish to use **sopf_TAssoc** in their implementations.

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

This class is not thread-safe. Even if you put semaphores around your calls to this class's methods, different tasks should not be setting the key and value. That situation is too prone to conflicts in setting the key or value correctly, with the result that the instance is in an unacceptable state for both tasks.

File Stem

tassoc

Base

sopf_MCollectible Class

Metaclass

SOMClass Class

Ancestor Classes

sopf_MCollectible Class
SOMObject Class

New Methods

sopfGetKey Method
sopfGetValue Method
sopfSetKey Method
sopfSetValue Method
sopfTAssocInitM Method
sopfTAssocInitMM Method

Overriding Methods

sopfDefaultInit Method
sopfDestruct Method

somfGetKey Method

Gets the key of an associated pair.

IDL Syntax

```
somf_MCollectible somfGetKey ( );
```

Description

The **somfGetKey** method obtains the key of the associated pair represented by the receiving object.

Note: You cannot override this method.

Parameters

receiver

A pointer to an object of class **somf_TAssoc**.

ev

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

Return Value

This method returns a pointer to the key of the associated pair.

Example

```
Environment *ev;
somf_TAssoc obj;
somf_MCollectible key;

ev = somGetGlobalEnvironment();

obj = somf_TAssocNew();

/* Add the key and value to obj */

/* Determine the key of obj */
key = _somfGetKey(obj, ev);

_somFree (obj);
```

Original Class

somf_TAssoc Class

Related Information

somfSetKey Method

somfSetValue Method

somfGetValue Method

somfGetValue Method

Gets the value to an associated (key, value) pair.

IDL Syntax

```
somf_MCollectible somfGetValue ( );
```

Description

The **somfGetValue** method gets the value to the associated pair represented by the receiving object.

Note: You cannot override this method.

Parameters

receiver

A pointer to an object of class **somf_TAssoc**.

ev

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

Return Value

This method returns a pointer to the value of the associated pair.

Example

```
Environment *ev;
somf_TAssoc obj;
somf_MCollectible value;

ev = somGetGlobalEnvironment();

obj = somf_TAssocNew();

/* Add the value and value to obj */

/* Determine the value of obj */
value = _somfGetValue(obj, ev);

_somFree (obj);
```

Original Class

somf_TAssoc Class

Related Information

somfSetValue Method
somfSetKey Method
somfGetKey Method

somfSetKey Method

Sets the key of an associated pair.

IDL Syntax

```
void somfSetKey (in somf_MCollectible k);
```

Description

The **somfSetKey** method sets the key of an associated pair represented by the receiving object.

Note: You cannot override this method.

Parameters

receiver

A pointer to an object of class **somf_TAssoc**.

ev

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

k

A pointer to an object of class **somf_MCollectible Class** which will be the key of the associated pair.

Example

```
Environment *ev;
somf_TAssoc obj;
somf_MCollectible key;

ev = somGetGlobalEnvironment();

obj = somf_TAssocNew();
key = somf_MCollectibleNew();

/* Add the key to obj */
_somfSetKey(obj, ev, key);

_somFree (obj);
_somFree (key);
```

Original Class

somf_TAssoc Class

Related Information

somfGetKey Method
somfSetValue Method
somfGetValue Method

somfSetValue Method

Sets the value to an associated (key, value) pair.

IDL Syntax

```
void somfSetValue (in somf_MCollectible v);
```

Description

The **somfSetValue** method sets the value to the associated pair represented by the receiving object.

Note: You cannot override this method.

Parameters

receiver

A pointer to an object of class **somf_TAssoc**.

ev

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

v

A pointer to an object of class **somf_MCollectible Class** which will be the value of the associated pair.

Example

```
Environment *ev;
somf_TAssoc obj;
somf_MCollectible value;

ev = somGetGlobalEnvironment();

obj = somf_TAssocNew();
value = somf_MCollectibleNew();

/* Add the value to obj */
_somfSetValue(obj, ev, value);

_somFree (obj);
_somFree (value);
```

Original Class

somf_TAssoc Class

Related Information

somfGetValue Method

somfGetKey Method

somfSetKey Method

somfTAssocInitM Method

Initializes a **somf_TAssoc** object to a given key (k). The value (v) is set to SOMF_NIL.

IDL Syntax

```
somf_TAssoc somfTAssocInitM (in somf_MCollectible k);
```

Description

The **somfTAssocInitM** method initializes an object of class **somf_TAssoc** to a given key (k). The value (v) is set to SOMF_NIL. An object of class **somf_TAssoc** is a pair.

Note: You cannot override this method.

Parameters

receiver

A pointer to an object of class **somf_TAssoc**.

ev

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

k

A pointer to an object of class **somf_MCollectible Class** that will be the key of the associated pair.

Return Value

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

Example

```
Environment *ev;
somf_TAssoc obj;

ev = somGetGlobalEnvironment();

obj = somf_TAssocNew();
_somfTAssocInitM(obj, ev, SOMF_NIL);

_somFree (obj);
```

Original Class

somf_TAssoc Class

Related Information

somfTAssocInitMM Method

somfTAssocInitMM Method

Initializes a **somf_TAssoc** object to a given key (k) and value (v).

IDL Syntax

```
somf_TAssoc somfTAssocInitMM (
    in somf_MCollectible k,
    in somf_MCollectible v);
```

Description

The **somfTAssocInitMM** method initializes an object of class **somf_TAssoc** to a given key (k) and value (v). An object of class **somf_TAssoc** is a pair.

Note: You cannot override this method.

Parameters

receiver

A pointer to an object of class **somf_TAssoc**.

ev

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

k

A pointer to an object of class **somf_MCollectible Class** that will be the key of the associated pair.

v

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

Return Value

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

Example

```
Environment *ev;
somf_TAssoc obj;

ev = somGetGlobalEnvironment();

obj = somf_TAssocNew();
_somfTAssocInitMM(obj, ev, SOMF_NIL, SOMF_NIL);

_somFree (obj);
```

Original Class

somf_TAssoc Class

Related Information

somfTAssocInitM Method

somf_TCollectibleLong Class

This class provides the user with a generic **somf_MCollectible** containing a long value.

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

This class is not thread-safe. Even if you put semaphores around your calls to this class's methods, you do not want different tasks setting the value. That situation is too prone to conflicts in setting the value correctly, with the result that the state of the instance is unacceptable for all but one task.

This class is reentrant.

File Stem

tclong

Base

somf_MCollectible Class

Metaclass

SOMClass Class

Ancestor Classes

somf_MCollectible Class
SOMObject Class

New Methods

somfGetValue Method
somfSetValue Method
somfTCollectibleLongInit Method

Overriding Methods

somDefaultInit Method
somflsEqual Method
somfHash Method

somfGetValue Method

Gets the value of the long in the receiving object.

IDL Syntax

```
long somfGetValue ( );
```

Description

The **somfGetValue** method gets the value of the long in the receiving object.

Parameters

receiver

A pointer to an object of class **somf_TCollectibleLong**.

ev

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

Return Value

Returns the value of the long.

Example

```
somf_TCollectibleLong l;
Environment *ev;

ev = somGetGlobalEnvironment();
l = somf_TCollectibleLongNew();
somPrintf("\n Value of l= %d\n", _somfGetValue(l,ev));

_somFree (l);
```

Original Class

somf_TCollectibleLong Class

Related Information

somfSetValue Method

somflsEqual Method

somfHash Method

Returns a value suitable for use as a hashing probe for the receiving object. Actually, it returns the value of the long.

IDL Syntax

```
long somfHash ( );
```

Description

The **somfHash** method returns a long value suitable for use as a hashing probe for the receiving object.

Parameters

receiver

A pointer to an object of class **somf_TCollectibleLong**.

ev

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

Return Value

This method returns the hash value for the receiving object.

Example

```
somf_TCollectibleLong l;
Environment *ev;

ev = somGetGlobalEnvironment();
l = somf_TCollectibleLongNew();
somPrintf("\n Hash Value of l= %d\n", _somfHash(l,ev));

_somFree (l);
```

Original Class

somf_MCollectible Class (overridden here)

Related Information

somfGetValue Method
somfSetValue Method
somfTCollectibleLongInit Method
somflsEqual Method

somflsEqual Method

Compares two objects and returns TRUE if a given *obj* is isomorphic to the receiving object.

IDL Syntax

```
boolean somflsEqual (in somf_MCollectible obj);
```

Description

The **somflsEqual** method returns TRUE if a specified object *obj* is isomorphic to the receiving object.

All of the utility classes allow you to specify what methods to use when comparing objects for insertion, deletion and so forth.

Parameters

receiver

A pointer to an object of class **somf_TCollectibleLong**.

ev

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

obj

A pointer to the **somf_MCollectible Class** object that the receiving object will be compared against.

Return Value

- TRUE, *obj* is equal to the receiving object.
- FALSE, *obj* is not equal to the receiving object.

Example

```
somf_TCollectibleLong l;
somf_TCollectibleLong ll;
Environment *ev;

ev = somGetGlobalEnvironment();
l = somf_TCollectibleLongNew();
ll = somf_TCollectibleLongNew();
_somfTCollectibleLongInit (ll, ev, 220);

/* if (*l == *ll) */
if (_somflsEqual (l, ev, ll))
    somPrintf("\n Why is l == ll?\n");
else
    somPrintf("\n l != ll\n");

_somFree (l);
_somFree (ll);
```

Original Class

somf_MCollectible Class (overridden here)

Related Information

somfGetValue Method
somfSetValue Method
somfTCollectibleLongInit Method

somfSetValue Method

Sets the value of a long in a **somf_TCollectibleLong** object.

IDL Syntax

```
void somfSetValue (in long v);
```

Description

The **somfSetValue** method sets the long value in an object of class **somf_TCollectibleLong**.

Parameters

receiver

A pointer to an object of class **somf_TCollectibleLong**.

ev

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

v

The value of the long.

Example

```
somf_TCollectibleLong l;
Environment *ev;

ev = somGetGlobalEnvironment();
l = somf_TCollectibleLongNew();

_somfSetValue(l, ev, 220);
somPrintf("\n Value of l= %d\n", _somfGetValue(l, ev));

_somFree (l);
```

Original Class

somf_TCollectibleLong Class

Related Information

somfGetValue Method

somflsEqual Method

somfTCollectibleLongInit Method

somfTCollectibleLongInit Method

Initializes a new object of class **somf_TCollectibleLong**.

IDL Syntax

```
somf_TCollectibleLong somfTCollectibleLongInit (in long v);
```

Description

The **somfTCollectibleLongInit** method initializes a new **somf_TCollectibleLong** object.

Note: You cannot override this method.

Parameters

receiver

A pointer to an object of class **somf_TCollectibleLong**.

ev

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

v

A pointer to the initial value of the **somf_TCollectibleLong**.

Return Value

This method returns a pointer to an initialized object of class **somf_TCollectibleLong**.

Example

```
somf_TCollectibleLong l;
Environment *ev;

ev = somGetGlobalEnvironment();
l = somf_TCollectibleLongNew();
_somfTCollectibleLongInit (l, ev, 44);

_somFree (l);
```

Original Class

somf_TCollectibleLong Class

Related Information

somfSetValue Method
somfGetValue Method
somfIsEqual Method
somfHash Method

somf_TCollection Class

This class represents a group of objects. It is implemented as an abstract class from which almost all main collection classes inherit methods.

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

When creating an unordered collection, your classes should inherit from **somf_TCollection**. When creating an ordered collection, your classes should inherit from **somf_TSequence Class**. The **somf_TCollection** class provides the pure virtual functions that constitute the framework for the methods that should be available in an unordered collection.

Note: The **somf_TCollection** class uses the **somflsEqual** 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 **somflsEqual Method** to equate entries, use the initialization methods to change to the **somflsSame Method**.

File Stem

tcollect

Base

somf_MCollectible Class

Metaclass

SOMClass Class

Ancestor Classes

somf_MCollectible Class
SOMObject Class

New Methods

somfAdd Method
somfAddAll Method
somfRemove Method
somfRemoveAll Method
somfDeleteAll Method
somfCount Method
somfMember Method
somfCreateIterator Method
somfTestFunction Method
somfSetTestFunction Method
somfTCollectionInit Method

Overriding Methods

somflsEqual Method

sopfAdd Method

Adds a specified *obj* to a collection.

IDL Syntax

```
sopf_MCollectible sopfAdd (in sopf_MCollectible obj);
```

Description

The **sopfAdd** method adds a specified object *obj* to the collection represented by the receiving object.

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

Parameters

receiver

A pointer to an object of class **sopf_TCollection**.

ev

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

obj

A pointer to an object of class **sopf_MCollectible** that will be added to the receiving object.

Return Value

- **sopf_MCollectible**, a pointer to the **sopf_MCollectible Class** object that had to be removed in order to add *obj*. (Recall that some of the main collection classes will only accept one occurrence of an object where the **sopfIsEqual Method** or **sopfIsSame Method** would be TRUE.)
- SOMF_NIL, no **sopf_MCollectible** had to be removed in order to add *obj*.

Example

You cannot use this method directly from this class; it must be overridden. If you invoke this method directly, an error message is written and processing will end. For examples of how this method looks when it is invoked, see **sopf_TDeque Class**, **sopf_TDictionary Class**, or any of the other classes that inherit from **sopf_TCollection**.

Original Class

sopf_TCollection Class

Related Information

sopfAddAll Method

somfAddAll Method

Adds all of the objects from a given collection into the receiving object.

IDL Syntax

```
void somfAddAll (in somf_TCollection col);
```

Description

The **somfAddAll** method adds all of the objects from a specified collection to the receiving object. Essentially, this is equivalent to passing in an iterator for the collection and then adding each element of the collection to the receiving object.

Parameters

receiver

A pointer to an object of class **somf_TCollection**.

ev

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

col

A pointer to an object of class **somf_TCollection**. All of the objects in the collection pointed to by *col* will be added to the receiving object.

Example

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

ev = somGetGlobalEnvironment();

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

/* Add All of the objects in s2 to s1 */
_somfAddAll(s1, ev, s2);

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

Original Class

somf_TCollection Class

Related Information

somfAdd Method

somfCount Method

Gets the number of objects in a collection.

IDL Syntax

```
long somfCount ( );
```

Description

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

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

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Thus, if any child of **somf_TCollection** is used with **somf_THashTable**, then the name of the method must 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 can reference this method as:

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

Parameters

receiver

A pointer to an object of class **somf_TCollection**.

ev

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

Return Value

This method returns a long indicating the number of objects in the receiving object.

Example

You cannot use this method directly from the **somf_TCollection** class; it must be overridden. If you invoke this method directly, an error message is written and processing will end. For examples of how this method looks when it is invoked, see **somf_TDeque Class** or **somf_TDictionary Class** or any of the other classes that inherit from **somf_TCollection**.

Original Class

somf_TCollection Class

somfCreateliterator Method

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

IDL Syntax

```
somf_Titerator somfCreateliterator ( );
```

Description

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

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

Parameters

receiver

A pointer to an object of class **somf_TCollection**.

ev

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

Return Value

This method returns a pointer to the new iterator.

Example

You cannot use this method directly from the **somf_TCollection** class; it must be overridden. If you invoke this method directly, an error message is written and processing will end. For examples of how this method looks when it is invoked, see **somf_TDeque Class** or **somf_TDictionary Class** or any of the other classes that inherit from **somf_TCollection**.

Original Class

somf_TCollection Class

somfDeleteAll Method

Removes all of the objects from the 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 receiving object and deallocates the storage that these objects might have owned (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 Method** to remove the objects from the collection and deleting them some other way.

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

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Thus, if any child of **somf_TCollection** is used with **somf_THashTable**, then the name of the method will have to be fully qualified (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 can reference this method as:

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

Parameters

receiver

A pointer to an object of class **somf_TCollection**.

ev

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

Example

You cannot use this method directly from the **somf_TCollection** class; it must be overridden. If you invoke this method directly, an error message is written and processing will end. For examples of how this method looks when it is invoked, see **somf_TDeque Class** or **somf_TDictionary Class** or any of the other classes that inherit from **somf_TCollection**.

Original Class

somf_TCollection Class

somflsEqual Method

Compares two objects and returns TRUE if a specified *obj* is isomorphic to the receiving object.

IDL Syntax

```
boolean somflsEqual (in somf_MCollectible obj);
```

Description

The **somflsEqual** method returns TRUE if a given object *obj* is isomorphic to the receiving object.

All of the utility classes allow you to specify what methods to use when comparing objects for insertion, deletion, and so forth.

Parameters

receiver

A pointer to an object of class **somf_TCollection**.

ev

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

obj

A pointer to the **somf_MCollectible Class** object that the receiving object will be compared against.

Return Value

- TRUE, *obj* is equal to the receiving object.
- FALSE, *obj* is not equal to the receiving object.

Original Class

somf_MCollectible Class (overridden here)

somfMember Method

Gets an *obj* in the collection.

IDL Syntax

```
somf_MCollectible somfMember (in somf_MCollectible obj);
```

Description

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

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Thus, if any child of **somf_TCollection** is used with **somf_THashTable**, then the name of the method will have to be fully qualified (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 can reference this method as:

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

Parameters

receiver

A pointer to an object of class **somf_TCollection**.

ev

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

obj

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

Return Value

- **somf_MCollectible**, a pointer to the object the method determined as the member.
- **SOMF_NIL**, indicates the object was not found.

Example

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

ev = somGetGlobalEnvironment();

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

/* See if obj is in dq */
if ( _somfMember(dq, ev, obj) != SOMF_NIL)
    somPrintf("\n obj is a Member\n");
else
    somPrintf("\n ERROR: obj should be a Member\n");

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

Original Class

somf_TCollection Class

sopfRemove Method

Removes an object from a collection.

IDL Syntax

```
sopf_MCollectible sopfRemove (in sopf_MCollectible obj);
```

Description

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

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

C cannot handle methods from different classes having the same name when they inherit the name from different parents. **sopfRemove** is a method name declared in multiple parents. You will probably have to fully qualify the method name. This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you can reference this method as:

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

Parameters

receiver

A pointer to an object of class **sopf_TCollection**.

ev

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

obj

A pointer to the object of class **sopf_MCollectible Class** to be removed from the collection.

Return Value

- **sopf_MCollectible**, a pointer to the object that was removed.
- **SOPF_NIL**, indicates the specified object was not found.

Example

You cannot use this method directly from the **sopf_TCollection** 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 **sopf_TDeque Class** or **sopf_TDictionary Class** or any of the other classes that inherit from **sopf_TCollection**.

Original Class

sopf_TCollection Class

Related Information

sopfRemoveAll Method

somfRemoveAll Method

Removes all of the objects from a collection.

IDL Syntax

```
void somfRemoveAll ( );
```

Description

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

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

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Thus, if any child of **somf_TCollection** is used with **somf_THashTable**, then the name of the method will have to be fully qualified (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 can reference this method as:

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

Parameters

receiver

A pointer to an object of class **somf_TCollection**.

ev

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

Example

You cannot use this method directly from the **somf_TCollection** class; it must be overridden. If you invoke this method directly, an error message is written and processing will end. For examples of how this method looks when it is invoked, see **somf_TDeque Class** or **somf_TDictionary Class** or any of the other classes that inherit from **somf_TCollection**.

Original Class

somf_TCollection Class

Related Information

somfRemove Method

somfSetTestFunction Method

Sets the test method for a collection.

IDL Syntax

```
void somfSetTestFunction (in somf_MCollectibleCompareFn testfn);
```

Description

The **somfSetTestFunction** method sets the test method to be used by the collection that is the receiving object.

Parameters

receiver

A pointer to an object of class **somf_TCollection**.

ev

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

testfn

A method pointer specifying either a **somflsEqual Method** or a **somflsSame Method**.

This argument should always be set to either

```
somf_MCollectibleClassData.somflsSame or  
somf_MCollectibleClassData.somflsEqual
```

because SOM needs a pointer to the original declaration of the method, which resides in **somf_MCollectible Class**. The **somf_TCollection** object will use this pointer to access the **somflsSame** or **somflsEqual** method that was declared and defined in the object being inserted into, or removed from, the **somf_TCollection** object.

Original Class

somf_TCollection Class

Related Information

somfTestFunction Method

somfTCollectionInit Method

Initializes a new object of class **somf_TCollection**.

IDL Syntax

```
somf_TCollection somfTCollectionInit (in somf_MCollectibleCompareFn testfn);
```

Description

The **somfTCollectionInit** method initializes a new object of class **somf_TCollection**.

Note: You cannot override this method.

Parameters

receiver

A pointer to an object of class **somf_TCollection**.

ev

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

testfn

A method pointer specifying either a **somflsEqual Method** or a **somflsSame Method**.

This argument should always be set to either

```
somf_MCollectibleClassData.somfIsSame or  
somf_MCollectibleClassData.somfIsEqual
```

because SOM needs a pointer to the original declaration of the method, which resides in **somf_MCollectible Class**. The **somf_TCollection** object will use this pointer to access the **somflsSame** or **somflsEqual** method that was declared and defined in the object being inserted into, or removed from, the **somf_TCollection** object.

Return Value

This method returns a pointer to an initialized object of class **somf_TCollection**.

Original Class

somf_TCollection Class

somfTestFunction Method

Determines the method that a collection uses for comparison testing.

IDL Syntax

```
somf_MCollectibleCompareFn somfTestFunction ( );
```

Description

The **somfTestFunction** method determines which method is used for comparison testing by the collection that is the receiving object. Comparison testing is performed on objects already contained in the collection and/or on objects being tested for eligibility.

Parameters

receiver

A pointer to an object of class **somf_TCollection**.

ev

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

Return Value

This method returns a pointer to the method that a **somf_TCollection** collection uses to compare two (existing or potential) objects of the collection. This test is usually either **somflsSame Method** or **somflsEqual Method**.

Original Class

somf_TCollection Class

Related Information

somfSetTestFunction Method

sopf_TDeque Class

A **sopf_TDeque** class is a child of **sopf_TSequence Class**. It is ordered based on the order in which objects are added to, or removed from, the collection. The **sopf_TDeque** class can also be used as a queue or a stack.

Each of these data structures are implemented in the **sopf_TDeque** class, with different methods processing the logically different structures. However, the **sopf_TDeque** class is more than all three data structures combined, because objects can be inserted and removed from any point in the **sopf_TDeque**. In addition, the **sopf_TDeque** is probably the most flexible of the data structures, because an object can appear in it more than once, and the only ordering in the data structure is determined by how elements are inserted into it.

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

Objects of the **sopf_MCollectible Class** that are inserted into a **sopf_TDeque** collection could override the **sopfIsSame** method.

Note: The **sopf_TDeque** class uses the **sopfIsSame** 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 do not want to use the **sopfIsSame** method to equate entries, use one of the initialization methods to change to the **sopfIsEqual** method. Just be aware that if the comparison methods are changed, the objects inserted into the **sopf_TDeque** must have **sopfIsEqual** and **sopfHash** overridden.

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 around to multiple threads, it is up to the code in those threads to guarantee thread-safe usage of the class.

File Stem

tdeq

Base

sopf_TSequence Class

Metaclass

SOPMClass Class

Ancestor Classes

sopf_TSequence Class
sopf_TCollection Class
sopf_MCollectible Class
SOPMObject Class

New Methods

sopfAddAfter Method
sopfAddBefore Method
sopfAddLast Method
sopfAddFirst Method
sopfRemoveLast Method
sopfRemoveFirst Method

somf_TDeque Class

- somfCreateSequenceIterator Method**
- somfRemoveQ Method**
- somfInsert Method**
- somfPop Method**
- somfPush Method**
- somfCreateNewLink Method**
- somfAssign Method**
- somfTDequeInitF Method**
- somfTDequeInitD Method**

Overriding Methods

- somDefaultInit Method**
- somDestruct Method**
- somfAdd Method**
- somfRemove Method**
- somfDeleteAll Method**
- somfRemoveAll Method**
- somfCount Method**
- somfAfter Method**
- somfBefore Method**
- somfLast Method**
- somfFirst Method**
- somfMember Method**
- somfCreateIterator Method**

somfAdd Method

Adds an object to a deque collection.

IDL Syntax

```
somf_MCollectible somfAdd (in somf_MCollectible obj);
```

Description

The **somfAdd** method adds a designated object *obj* to the deque collection represented by the receiving object.

Parameters

receiver

A pointer to an object of class **somf_TDeque**.

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 that was added.

Example

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

ev = somGetGlobalEnvironment();

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

_somfAdd(dq, ev, obj);

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

Original Class

somf_TCollection Class (overridden here)

Related Information

somfAddAfter Method
somfAddBefore Method
somfAddFirst Method
somfAddLast Method

somfAddAfter Method

Adds a new object to a deque collection after a specified existing object.

IDL Syntax

```
void somfAddAfter (
    in somf_MCollectible existingobj,
    in somf_MCollectible tobeadded);
```

Description

The **somfAddAfter** method adds the designated new object, *tobeadded*, to the deque collection after the specified existing object, *existingobj*.

Parameters

receiver

A pointer to an object of class **somf_TDeque**.

ev

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

existingobj

A pointer to the existing **somf_MCollectible** object after which the new object will be added.

tobeadded

A pointer to the new **somf_MCollectible** object to be added to the deque.

Example

```
somf_TDeque dq;
<your Class which inherits from somf_MCollectible> obj1;
<your Class which inherits from somf_MCollectible> obj2;
Environment *ev;

ev = somGetGlobalEnvironment();

dq = somf_TDequeNew();
obj1 = <your Class which inherits from somf_MCollectible>New();
obj2 = <your Class which inherits from somf_MCollectible>New();

_somfAddFirst(dq, ev, obj1);
_somfAddAfter(dq, ev, obj1, obj2);

_somFree (dq);
_somFree (obj1);
_somFree (obj2);
```

Original Class

somf_TDeque Class

Related Information

somfAdd Method
somfAddBefore Method
somfAddFirst Method
somfAddLast Method

somfAddBefore Method

Adds a new object to a deque collection before a specified existing object.

IDL Syntax

```
void somfAddBefore (
    in somf_MCollectible existobj,
    in somf_MCollectible tobeadded);
```

Description

The **somfAddBefore** method adds the designated new object, *tobeadded*, to the deque collection before the specified existing object, *existobj*.

Parameters

receiver

A pointer to an object of class **somf_TDeque**.

ev

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

existobj

A pointer to the existing **somf_MCollectible** object before which the new object will be added.

tobeadded

A pointer to the new **somf_MCollectible** object to be added to the deque.

Example

```
somf_TDeque dq;
<your Class which inherits from somf_MCollectible> obj1;
<your Class which inherits from somf_MCollectible> obj2;
Environment *ev;

ev = somGetGlobalEnvironment();

dq = somf_TDequeNew();
obj1 = <your Class which inherits from somf_MCollectible>New();
obj2 = <your Class which inherits from somf_MCollectible>New();

_somfAddFirst(dq, ev, obj1);
_somfAddBefore(dq, ev, obj1, obj2);

_somFree (dq);
_somFree (obj1);
_somFree (obj2);
```

Original Class

somf_TDeque Class

Related Information

somfAdd Method
somfAddAfter Method
somfAddFirst Method
somfAddLast Method

somfAddFirst Method

Adds a new object as the first object in a deque collection.

IDL Syntax

```
void somfAddFirst (in somf_MCollectible obj);
```

Description

The **somfAddFirst** method adds the designated new object *obj* as the first object in the deque collection represented by the receiving object.

Parameters

receiver

A pointer to an object of class **somf_TDeque**.

ev

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

obj

A pointer to the object of class **somf_MCollectible** to be added to the deque.

Example

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

ev = somGetGlobalEnvironment();

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

_somfAddFirst(dq, ev, obj);

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

Original Class

somf_TDeque Class

Related Information

somfAdd Method

somfAddAfter Method

somfAddBefore Method

somfAddLast Method

somfAddLast Method

Adds a new object as the last object in a deque collection.

IDL Syntax

```
void somfAddLast (in somf_MCollectible obj);
```

Description

The **somfAddLast** method adds the new object *obj* as the last object in the deque collection.

Parameters

receiver

A pointer to an object of class **somf_TDeque**.

ev

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

obj

A pointer to the object of class **somf_MCollectible** to be added to the deque.

Example

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

ev = somGetGlobalEnvironment();

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

_somfAddLast(dq, ev, obj);

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

Original Class

somf_TDeque Class

Related Information

somfAdd Method
somfAddAfter Method
somfAddBefore Method
somfAddFirst Method

somfAfter Method

Gets the object found after a specified object in a deque collection.

IDL Syntax

```
somf_MCollectible somfAfter (in somf_MCollectible obj);
```

Description

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

Parameters

receiver

A pointer to an object of class **somf_TDeque**.

ev

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

obj

A pointer to **thesomf_MCollectible** that is in front of the returned *obj*.

Return Value

- **somf_MCollectible**, a pointer to the object of class **somf_MCollectible** that comes after *obj*.
- SOMF_NIL, *obj* is the last object in the collection or could not be found.

Example

```
somf_TDeque dq;
somf_MCollectible obj;
Environment *ev;
<Your Class which inherits from somf_MCollectible> a1;

ev = somGetGlobalEnvironment();

dq = somf_TDequeNew();
a1 = <Your Class which inherits from somf_MCollectible>New();

/* Add some objects to dq */

/* set obj to point to the object after a1 */
obj = _somfAfter(dq, ev, a1);

_somFree (dq);
_somFree (a1);
```

Original Class

somf_TSequence Class (overridden here)

Related Information

somfBefore Method

somfFirst Method

somfLast Method

somfAssign Method

Assigns a deque collection as being equal to a given source deque.

IDL Syntax

```
void somfAssign (in somf_TDeque s);
```

Description

The **somfAssign** method assigns the deque receiving object to be equal to the source deque object. That is, the method sets or resets the instance variables of the receiver to the values of the source. This operation is logically equivalent to using the equal (=) 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_TCollection** is used with **somf_THashTable**, then the name of the method will have to be fully qualified (example: **somf_TDeque_somfAssign**). This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you can reference this method as:

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

Parameters

receiver

A pointer to an object of class **somf_TDeque**.

ev

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

s

A pointer to the **somf_TDeque** object to which the receiving object will be set equal.

Example

```
somf_TDeque dq1;
somf_TDeque dq2;
Environment *ev;

ev = somGetGlobalEnvironment();

dq1 = somf_TDequeNew();
dq2 = somf_TDequeNew();

/* Add som objects to dq1 */

/* Assign dq2 = dq1 */
somf_TDeque_somfAssign(dq2, ev, dq1);

_somFree (dq1);
_somFree (dq2);
```

Original Class

somf_TDeque Class

somfBefore Method

Gets the object found before a specified object in a deque collection.

IDL Syntax

```
somf_MCollectible somfBefore (in somf_MCollectible obj);
```

Description

The **somfBefore** method returns the object found immediately before the designated object *obj* in a deque collection represented by the receiving object.

Parameters

receiver

A pointer to an object of class **somf_TDeque**.

ev

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

obj

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

Return Value

- **somf_MCollectible**, a pointer to the **somf_MCollectible** object that precedes *obj*.
- SOMF_NIL, the *obj* is the first object in the receiving object or could not be found.

Example

```
somf_TDeque dq;
somf_MCollectible obj;
Environment *ev;
<Your Class which inherits from somf_MCollectible> a1;

ev = somGetGlobalEnvironment();

dq = somf_TDequeNew();
a1 = <Your Class which inherits from somf_MCollectible>New();

/* Add some objects to dq */

/* set obj to point to the object before a1 */
obj = _somfBefore(dq, ev, a1);

_somFree (dq);
_somFree (a1);
```

Original Class

somf_TSequence Class (overridden here)

Related Information

somfAfter Method

somfFirst Method

somfLast Method

somfCount Method

Gets the number of objects in a deque collection.

IDL Syntax

```
long somfCount ( );
```

Description

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

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

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

Parameters

receiver

A pointer to an object of class **somf_TDeque**.

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_TDeque dq;
Environment *ev;

ev = somGetGlobalEnvironment();

dq = somf_TDequeNew();

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

_somFree (dq);
```

Original Class

somf_TCollection Class (overridden here)

somfCreateliterator Method

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

IDL Syntax

```
somf_Titerator somfCreateliterator ( );
```

Description

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

Note: This is one of three ways to initialize a **somf_TDequeliterator Class** to point to an instance of a **somf_TDeque**. One other way is to use the **somf_TDequeliterator** initializer method, and the final way is to use **somfCreateSequenceliterator Method**.

Parameters

receiver

A pointer to an object of class **somf_TDeque**.

ev

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

Return Value

This method returns a pointer to the new iterator.

Example

```
somf_TDeque dq;
Environment *ev;
somf_TDequeIterator itr;

ev = somGetGlobalEnvironment();

dq = somf_TDequeNew();
itr = (somf_TDequeIterator*) _somfCreateIterator(dq, ev);

_somFree (dq);
_somFree (itr);
```

Original Class

somf_TCollection Class (overridden here)

Related Information

somfCreateSequenceliterator Method

somfCreateNewLink Method

Creates a new link in a **somf_TDeque** collection, given two objects as the previous and next **somf_TDequeLinkable Class** links, and the value of the new link.

IDL Syntax

```
somf_TDequeLinkable somfCreateNewLink (
    in somf_TDequeLinkable p,
    in somf_TDequeLinkable n,
    in somf_MCollectible v);
```

Description

The **somfCreateNewLink** method creates a new link in a **somf_TDeque** collection, given the previous and next **somf_TDequeLinkable** objects, as well as the value of the new link.

When inheriting from this class, this method can be overridden if you want to customize how a **somf_TDeque** object creates a new link in your derived class.

Parameters

receiver

A pointer to an object of class **somf_TDeque**.

ev

A pointer to the Environment structure for the calling method.

p

A pointer to the **somf_TDequeLinkable** object before this one.

n

A pointer to the **somf_TDequeLinkable** object after this one.

v

A pointer to the **somf_MCollectible** object that represents the value of the new **somf_TDequeLinkable**.

Return Value

This method returns a pointer to the new **somf_TDequeLinkable** object.

Original Class

somf_TDeque Class

somfCreateSequenceliterator Method

Returns a new iterator that is suitable for iterating over the objects in the given deque collection.

IDL Syntax

```
somf_TSequenceliterator somfCreateSequenceliterator ( );
```

Description

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

Note: This is one of three ways to initialize a **somf_TDequeliterator Class** to point to an instance of a **somf_TDeque**. One other way is to use **somf_TDequeliterator**'s initializer method, and the final way is to use the **somf_TDeque**'s **somfCreateliterator Method**.

This method is identical to **somfCreateliterator**; you could use either one. The only difference is that the type of the return value for this method is a **somf_TSequenceliterator Class**, and the return type for the return value of **somfCreateliterator** is a **somf_Titerator Class**. However, both methods return an instance of a **somf_TDequeliterator** that has been typed correctly.

Parameters

receiver

A pointer to an object of class **somf_TDeque**.

ev

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

Return Value

This method returns a pointer to the new iterator.

Example

```
somf_TDeque dq;
Environment *ev;
somf_TDequeIterator itr;

ev = somGetGlobalEnvironment();

dq = somf_TDequeNew();
itr = (somf_TDequeIterator*) _somfCreateSequenceIterator(dq, ev);

_somFree (dq);
_somFree (itr);
```

Original Class

somf_TDeque Class

Related Information

somfCreateliterator Method

somfDeleteAll Method

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

IDL Syntax

```
void somfDeleteAll ( );
```

Description

The **somfDeleteAll** method removes all of the objects from the deque 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 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 Method** 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_TDeque_somfDeleteAll**). This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you can reference this method as:

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

Parameters

receiver

A pointer to an object of class **somf_TDeque**.

ev

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

Example

```
somf_TDeque dq;
Environment *ev;

ev = somGetGlobalEnvironment();

dq = somf_TDequeNew();

/* Add some objects to dq */

/* Remove all objects in dq AND DELETE ALL INSTANCES */
_somfDeleteAll(dq, ev);

_somFree (dq);
```

Original Class

somf_TCollection Class (overridden here)

somfFirst Method

Gets the first object in a deque collection.

IDL Syntax

```
somf_MCollectible somfFirst ();
```

Description

The **somfFirst** method determines the first object in the deque collection represented by the receiving object, and returns a pointer to the object (if found).

C cannot handle methods from different classes having the same name when they inherit the name from different parents. **somfFirst** is a method name declared in multiple parents (example: **somf_TSequence**, **somf_TIterator** and so forth.). You will probably have to fully qualify the method name (example: **somf_TDeque_somfFirst**). This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you can reference this method as:

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

Parameters

receiver

A pointer to an object of class **somf_TDeque**.

ev

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

Return Value

- **somf_MCollectible**, a pointer to the first **somf_MCollectible** object in the collection.
- **SOMF_NIL**, nothing is in the collection.

Example

```
somf_TDeque dq;
Environment *ev;
somf_MCollectible first;

ev = somGetGlobalEnvironment();

dq = somf_TDequeNew();

/* Add some objects to dq */

first = somf_TDeque_somfFirst(dq, ev);
/* do something with the first object in the somf_TDeque */

_somFree (dq);
```

Original Class

somf_TSequence Class (overridden here)

Related Information

somfAddAfter Method
somfBefore Method
somfLast Method

somfInsert Method

Adds an object to the end of the deque/queue.

IDL Syntax

```
void somfInsert (in somf_MCollectible obj);
```

Description

The **somfInsert** method appends the object *obj* to the end of the deque/queue represented by the receiving object.

This method can be used with **somfRemoveQ Method** to simulate a queue.

Parameters

receiver

A pointer to an object of class **somf_TDeque**.

ev

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

obj

A pointer to the **somf_MCollectible** object to be added to the deque.

Example

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

ev = somGetGlobalEnvironment();

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

_somfInsert(dq, ev, obj);

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

Original Class

somf_TDeque Class

Related Information

somfRemoveQ Method

somfLast Method

Gets the last object in a given deque collection.

IDL Syntax

```
somf_MCollectible somfLast( );
```

Description

The **somfLast** method gets the last object in the deque collection 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_TSequence** is used with a child of **somf_TSequenceliterator** or with **somf_TPrimitiveLinkedListIterator**, then the name of the method must be fully qualified (for example: **somf_TDeque_somfLast**). This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you can reference this method as:

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

Parameters

receiver

A pointer to an object of class **somf_TDeque**.

ev

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

Return Value

- **somf_MCollectible**, a pointer to the last **somf_MCollectible** object in the collection.
- **SOMF_NIL**, nothing is in the collection.

Example

```
somf_TDeque dq;
Environment *ev;
somf_MCollectible last;

ev = somGetGlobalEnvironment();

dq = somf_TDequeNew();

/* Add some objects to dq */

last = somf_TDeque_somfLast(dq, ev);
/* do something with the last object in the somf_TDeque */

_somFree (dq);
```

Original Class

somf_TSequence Class (overridden here)

Related Information

somfAfter Method
somfBefore Method
somfFirst Method

somfMember Method

Gets an object in a deque collection.

IDL Syntax

```
somf_MCollectible somfMember (in somf_MCollectible obj);
```

Description

The **somfMember** method determines whether object `obj` is in the deque collection represented by the receiving object, and returns a pointer to the object (if found).

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

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

Parameters

receiver

A pointer to an object of class **somf_TDeque**.

ev

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

obj

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

Return Value

- **somf_MCollectible**, a pointer to the object the method determined as the member.
- **SOMF_NIL**, `obj` was not found.

Example

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

ev = somGetGlobalEnvironment();

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

/* See if obj is in dq */
if ( _somfMember(dq, ev, obj) != SOMF_NIL)
    somPrintf("\n obj is a Member\n");
else
    somPrintf("\n ERROR: obj should be a Member\n");

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

Original Class

somf_TCollection Class (overridden here)

somfPop Method

Removes the object on top of a deque/stack.

IDL Syntax

```
somf_MCollectible somfPop ( );
```

Description

The **somfPop** method removes the object on top of the deque/stack represented by the receiving object.

Note: This call can be used to simulate a stack.

Parameters

receiver

A pointer to an object of class **somf_TDeque**.

ev

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

Return Value

This method returns a pointer to the **somf_MCollectible** object removed from the deque or stack. Or. SOMF_NIL is returned if the collection is empty.

Example

```
somf_TDeque dq;
somf_MCollectible obj;
Environment *ev;

ev = somGetGlobalEnvironment();

dq = somf_TDequeNew();

/* Use _somfPush to push objects onto the stack */

/* Pop an object from the stack */
obj = _somfPop(dq, ev);

_somFree (dq);
```

Original Class

somf_TDeque Class

Related Information

somfPush Method

somfPush Method

Adds an object to the top of a deque/stack.

IDL Syntax

```
void somfPush (in somf_MCollectible obj);
```

Description

The **somfPush** method adds the specified object *obj* to the top of the deque or stack represented by the receiving object.

Note: This call can be used to simulate a stack.

Parameters

receiver

A pointer to an object of class **somf_TDeque**.

ev

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

obj

A pointer to the **somf_MCollectible** object to be added to the deque.

Example

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

ev = somGetGlobalEnvironment();

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

_somfPush(dq, ev, obj);

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

Original Class

somf_TDeque Class

Related Information

somfPop Method

somfRemove Method

Removes an object from a deque collection.

IDL Syntax

```
somf_MCollectible somfRemove (in somf_MCollectible obj);
```

Description

The **somfRemove** method removes the object *obj* from the deque collection 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 (example: **somf_TCollection**, **somf_THashTable**, **somf_TIterator**). You will probably have to fully qualify the method name. This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you can reference this method as:

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

Parameters

receiver

A pointer to an object of class **somf_TDeque**.

ev

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

obj

A pointer to the **somf_MCollectible** object to be removed from the deque collection.

Return Value

- **somf_MCollectible**, a pointer to the object that was removed.
- **SOMF_NIL**, the object was not found.

Example

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

ev = somGetGlobalEnvironment();

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

/* Add some values to dq */

somf_TDeque_somfRemove(dq, ev, obj);

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

Original Class

somf_TCollection Class (overridden here)

Related Information

somfRemoveAll Method
somfRemoveFirst Method
somfRemoveLast Method

somfRemoveAll Method

Removes all of the objects from a deque collection.

IDL Syntax

```
void somfRemoveAll ( );
```

Description

The **somfRemoveAll** method removes all of the objects from the deque collection 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 must be fully qualified. This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you can reference this method as:

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

Parameters

receiver

A pointer to an object of class **somf_TDeque**.

ev

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

Example

```
somf_TDeque dq;
Environment *ev;

ev = somGetGlobalEnvironment();

dq = somf_TDequeNew();

/* Add some objects to dq */

/* Remove all of the objects in dq */
_somfRemoveAll (dq, ev);

_somFree (dq);
```

Original Class

somf_TCollection Class (overridden here)

Related Information

somfRemove Method
somfRemoveFirst Method
somfRemoveLast Method

somfRemoveFirst Method

Removes the first object in a deque collection.

IDL Syntax

```
somf_MCollectible somfRemoveFirst ( );
```

Description

The **somfRemoveFirst** method determines the first object in the deque collection represented by the receiving object, and removes it.

Parameters

receiver

A pointer to an object of class **somf_TDeque**.

ev

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

Return Value

- **somf_MCollectible**, a pointer to the **somf_MCollectible** object that was removed.
- **SOMF_NIL**, the collection is empty.

Example

```
somf_TDeque dq;
Environment *ev;

ev = somGetGlobalEnvironment();

dq = somf_TDequeNew();

/* Add some objects to dq */

if (_somfRemoveFirst(dq, ev) == SOMF_NIL)
    somPrintf(
        " ERROR: The first object should have been removed.\n");

_somFree (dq);
```

Original Class

somf_TDeque Class

Related Information

somfRemove Method
somfRemoveAll Method
somfRemoveLast Method

somfRemoveLast Method

Removes the last object in a deque collection.

IDL Syntax

```
somf_MCollectible somfRemoveLast ( );
```

Description

The **somfRemoveLast** method determines the last object in the deque collection represented by the receiving object, and removes it.

Parameters

receiver

A pointer to an object of class **somf_TDeque**.

ev

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

Return Value

- **somf_MCollectible**, a pointer to the **somf_MCollectible** object that was removed.
- **SOMF_NIL**, the collection is empty.

Example

```
somf_TDeque dq;
Environment *ev;

ev = somGetGlobalEnvironment();

dq = somf_TDequeNew();

/* Add some objects to dq */

if (_somfRemoveLast(dq, ev) == SOMF_NIL)
    somPrintf(
        " ERROR: The last object should have been removed.\n");

_somFree (dq);
```

Original Class

somf_TDeque Class

Related Information

somfRemove Method

somfRemoveAll Method

somfRemoveFirst Method

somfRemoveQ Method

Removes the first object from a deque/queue.

IDL Syntax

```
somf_MCollectible somfRemoveQ ( );
```

Description

The **somfRemoveQ** method removes the first object from the deque/queue represented by the receiving object.

Note: This method can be used with **somfInsert** to simulate a queue.

Parameters

receiver

A pointer to an object of class **somf_TDeque**.

ev

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

Return Value

- **somf_MCollectible**, a pointer to the object that was removed.
- **SOMF_NIL**, the object was not found.

Example

```
somf_TDeque dq;
somf_MCollectible obj;
Environment *ev;

ev = somGetGlobalEnvironment();

dq = somf_TDequeNew();

/* Use _somfInsert to insert objects into the queue */

/* Remove an object from the queue */
obj = _somfRemoveQ(dq, ev);

_somFree (dq);
```

Original Class

somf_TDeque Class

Related Information

somfInsert Method

somfTDequeInitD Method

Initializes a new deque, setting it equal to a given **somf_TDeque** source object.

IDL Syntax

```
somf_TDeque somfTDequeInitD (in somf_TDeque s);
```

Description

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

Note: You cannot override this method.

Parameters

receiver

A pointer to an object of class **somf_TDeque**.

ev

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

s

A pointer to the source deque to which the receiving object will be set equal.

Return Value

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

Example

```
somf_TDeque dq1;
somf_TDeque dq2;
Environment *ev;

ev = somGetGlobalEnvironment();

dq1 = somf_TDequeNew();
dq2 = somf_TDequeNew();
_somfTDequeInitD(dq2, ev, dq1);

_somFree (dq1);
_somFree (dq2);
```

Original Class

somf_TDeque Class

Related Information

somfTDequeInitF Method

somfTDequeInitF Method

Initializes a new deque collection, specifying the comparison method that it will use.

IDL Syntax

```
somf_TDeque somfTDequeInitF (in somf_MCollectibleCompareFn testfn);
```

Description

The **somfTDequeInitF** method initializes a new deque represented by the receiving object. The method also establishes the comparison method that the new deque will use to compare current/potential objects for the collection, as determined by the *testfn* argument.

Note: You cannot override this method.

Parameters

receiver

A pointer to an object of class **somf_TDeque**.

ev

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

testfn

A method pointer specifying either a **somflsEqual** or **somflsSame** method.

This argument should always be set to either

```
somf_MCollectibleClassData.somflsSame or  
somf_MCollectibleClassData.somflsEqual.
```

This specification is necessary because SOM needs a pointer to the original declaration of the method, residing in **somf_MCollectible Class**. The **somf_TDeque** object will use this pointer to access the **somflsSame Method** or **somflsEqual Method** that was declared and defined in the object being inserted into, or removed from, the **somf_TDeque** object.

Return Value

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

Example

```
somf_TDeque dq1;  
Environment *ev;  
  
ev = somGetGlobalEnvironment();  
  
dq1 = somf_TDequeNew();  
_somfTDequeInitF(dq1, ev,  
                 somf_MCollectibleClassData.somflsSame);  
  
_somFree (dq1);
```

Original Class

somf_TDeque Class

Related Information

somfTDequeInitD Method

somf_TDequelterator Class

An iterator for the **somf_TDeque Class** that will iterate over all of the objects in a deque.

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

File Stem

tdeqitr

Base

somf_TSequenceliterator Class

Metaclass

SOMClass

Ancestor Classes

somf_TSequenceliterator Class

somf_TIterator Class

SOMObject Class

New Methods

somfTDequelteratorInit Method

Overriding Methods

somfFirst Method

somfNext Method

somfLast Method

somfPrevious Method

somfRemove Method

somfFirst Method

Resets the iterator and returns the first object from a deque collection.

IDL Syntax

```
somf_MCollectible somfFirst ();
```

Description

The **somfFirst** method resets the iterator and returns the first object in the deque collection that corresponds to the deque iterator represented by the receiving object. This resets the iterator to the beginning of the collection even if other operations on the collection cause the iterator to be invalidated. In the second case, this 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, and you will probably have to fully qualify it. This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you can reference this method as:

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

Parameters

receiver

A pointer to an object of class **somf_TDequeIterator**.

ev

A pointer to the Environment structure for the calling method.

Return Value

- A pointer to the first **somf_MCollectible** object in the collection.
- SOMF_NIL is returned if the collection is empty.

Example

```
somf_TDeque dq;
Environment *ev;
somf_TDequeIterator itr;
somf_MCollectible itrobj;

ev = somGetGlobalEnvironment();
dq = somf_TDequeNew();
itr = somf_TDequeIteratorNew();
_somfTDequeIteratorInit(itr, ev, dq);

/* Add some object to dq */
/* Iterate through the TDeque */
itrobj = somf_TDequeIterator_somfFirst(itr, ev);
while (itrobj != SOMF_NIL)
{
    /* Do something with itrobj */
    itrobj = _somfNext(itr, ev);
}
_somFree (dq);
_somFree (itr);
```

Original Class

somf_TIterator Class (overridden here)

Related Information

somfNext Method

somfLast Method

Gets the last object in the deque collection.

IDL Syntax

```
somf_MCollectible somfLast ( );
```

Description

The **somfLast** method determines the last object in the deque collection that corresponds to the iterator represented by the receiving object, and returns a pointer to the last object (if found).

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**). You will probably have to fully qualify the name of the method. This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you can reference this method as:

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

Parameters

receiver

A pointer to an object of class **somf_TDequeIterator**.

ev

A pointer to the Environment structure for the calling method.

Return Value

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

Example

```
somf_TDeque dq;
somf_TDequeIterator itr;
somf_MCollectible itrobj;
Environment *ev;

ev = somGetGlobalEnvironment();

dq = somf_TDequeNew();
itr = somf_TDequeIteratorNew();
_somfTDequeIteratorInit(itr, ev, dq);

/* Add some objects to dq */

/* set obj to point to the last object in dq */
itrobj = somf_TDequeIterator_somfLast(itr, ev);

_somFree (dq);
_somFree (itr);
```

Original Class

somf_TSequenceIterator Class (overridden here)

Related Information

somfPrevious Method

somfNext Method

Gets the next object in a deque collection.

IDL Syntax

```
somf_MCollectible somfNext ();
```

Description

The **somfNext** method determines the next object in the deque collection that corresponds to the deque iterator represented by the receiving object. The method also returns a pointer to the next object, if found. Objects are retrieved in an order that reflects the “ordered-ness” of the collection, or the lack of ordering on the collection 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 **somf_TPrimitiveLinkedListIterator**, then the name of the method must be fully qualified. This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you can reference this method as:

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

If the **somf_TDeque Class** collection has changed (other than through the use of the **somfRemove Method** of this iterator) since the last time the **somfFirst Method** or **somfLast Method** was called, the iterator becomes invalid and will *fail* when asked to find the next object. If the collection’s **somfAdd Method** were called after starting to iterate through the collection, the iterator 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.

Parameters

receiver

A pointer to an object of class **somf_TDequeIterator**.

ev

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

Return Value

- **somf_MCollectible**, a pointer to the next **somf_MCollectible Class** object in the collection.
- **SOMF_NIL**, the end of the collection has been reached.

Example

```
somf_TDeque dq;
Environment *ev;
somf_TDequeIterator itr;
somf_MCollectible itrobj;

ev = somGetGlobalEnvironment();

dq = somf_TDequeNew();
itr = somf_TDequeIteratorNew();
_somfTDequeIteratorInit(itr, ev, dq);

/* Add some object to dq */

/* Iterate through the TDeque */
itrobj = somf_TDequeIterator_somfFirst(itr, ev);
while (itrobj != SOMF_NIL)
{
```

```
        /* Do something with itrobj */  
        itrobj = _somfNext(itr, ev);  
    }  
  
    _somFree (dq);  
    _somFree (itr);
```

Original Class

somf_Tliterator Class (overridden here)

Related Information

somfFirst Method

sopfPrevious Method

Gets the previous object in a deque collection.

IDL Syntax

```
sopf_MCollectible sompPrevious ( );
```

Description

The **sopfPrevious** method determines the previous object in the deque collection that corresponds to the deque iterator represented by the receiving object. The method returns a pointer to the previous object, if found.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. If any child of **sopf_TSequenceliterator** is used with **sopf_TPrimitiveLinkedListiterator**, then the name of the method must be fully qualified so the linker can tell them apart. This is not a problem in C++, for you can reference the method as:

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

If the **sopf_TDeque Class** collection changes while using this iterator, it becomes invalid and will fail to find the previous object. If the collection's **sopfAdd Method** is called after starting to iterate through the collection, it will not allow iteration to continue. The iterator must be reset, and the easiest way is to call the iterator's **sopfLast Method**.

Parameters

receiver

A pointer to an object of class **sopf_TDequeiterator**.

ev

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

Return Value

- **sopf_MCollectible**, a pointer to the previous **sopf_MCollectible Class** object in the collection.
- **SOMF_NIL**, the beginning of the collection has been reached.

Example

```
sopf_TDeque dq;
sopf_TDequeIterator itr;
sopf_MCollectible itrobj;
Environment *ev;
ev = somGetGlobalEnvironment();
dq = somp_TDequeNew();
itr = somp_TDequeIteratorNew();
_sopfTDequeIteratorInit(itr, ev, dq);
/* Add some objects to dq */
/* set itrobj to point to the next to last object in dq */
sopf_TDequeIterator_sopfLast(itr, ev);
itrobj = _sopfPrevious(itr, ev);
_sopfFree (dq);
_sopfFree (itr);
```

Original Class

sopf_TSequenceliterator Class (overridden here)

Related Information

sopfLast Method

somfRemove Method

Removes the current object from a deque collection.

IDL Syntax

```
void somfRemove ();
```

Description

The **somfRemove** method removes the current object from the deque collection that corresponds to the deque iterator represented by the receiving object. **somfRemove** is the only way to remove an object from a collection during iteration. If multiple iterators are in process, all the other iterators are invalidated.

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. You will probably have to fully qualify the method name so the linker can tell them apart. This is not a problem in C++. In C++ you can reference this method as:

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

If the **somf_TDeque Class** collection has changed since the last time the **somfFirst Method** or **somfLast Method** was called, the iterator becomes invalid and will fail if asked to remove an object. If the collection's **somfAdd Method** were called after starting to iterate through the collection, it would not allow iteration to continue. The iterator must be reset, and the easiest way is to call the iterator's **somfFirst** or **somfLast** method and restart.

Parameters

receiver

A pointer to an object of class **somf_TDequeIterator**.

ev

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

Example

```
somf_TDeque dq;
Environment *ev;
somf_TDequeIterator itr;
somf_MCollectible itrobj;

ev = somGetGlobalEnvironment();

dq = somf_TDequeNew();
itr = somf_TDequeIteratorNew();
_somfTDequeIteratorInit(itr, ev, dq);

/* Add some objects to dq */
/* Use the Iterator's Remove to remove the first object */
itrobj = somf_TDequeIterator_somfFirst(itr, ev);
somf_TDequeIterator_somfRemove(itr, ev);

_somFree (dq);
_somFree (itr);
```

Original Class

somf_TIterator Class (overridden here)

somfTDequelteratorInit Method

Initializes a **somf_TDequelterator** iterator for a deque collection.

IDL Syntax

```
somf_TDequelterator somfTDequelteratorInit (in somf_TDeque h);
```

Description

The **somfTDequelteratorInit** method initializes an iterator of class **somf_TDequelterator**, given the **somf_TDeque Class** collection over which iteration is needed.

This is one of three ways to initialize a **somf_TDequelterator** to point to an instance of a **somf_TDeque** collection. One other way is to use the **somfCreateliterator Method** of the **somf_TDeque** class, and the final way is to use the **somf_TDeque** class's **somfCreateSequenceliterator Method**.

Note: You cannot override this method

Parameters

receiver

A pointer to an object of class **somf_TDequelterator**.

ev

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

h

A pointer to the deque object that the receiving object will iterate over.

Return Value

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

Example

```
somf_TDeque dq;
Environment *ev;
somf_TDequeIterator itr;

ev = somGetGlobalEnvironment();

dq = somf_TDequeNew();
itr = somf_TDequeIteratorNew();
_somfTDequeIteratorInit(itr, ev, dq);

_somFree (dq);
_somFree (itr);
```

Original Class

somf_TDequelterator

sopf_TDequeLinkable Class

The **sopf_TDequeLinkable** class is a subclass of **sopf_MLinkable Class**. It provides a generic **sopf_MLinkable** class to contain **sopf_MCollectible Class**. An object of class **sopf_TDequeLinkable** is used (transparently) by the **sopf_TDeque Class** for each node of a deque collection. The **sopf_TDequeLinkable** object provides the linkability (the left and right links) to its two adjacent nodes in the collection.

The **sopf_TDequeLinkable** class and methods will probably be of interest to programmers only in two situations:

- if you are creating a new class that needs linkable nodes between objects of the class
- if you are creating a new class that inherits from **sopf_TDeque**, and it would be appropriate to override some methods of the **sopf_TDequeLinkable** class to define additional functionality for those methods.

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

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

This class is reentrant.

File Stem

tdeqlink

Base

sopf_MLinkable Class

Metaclass

SOMClass

Ancestor Classes

sopf_MLinkable Class
SOMObject

New Methods

sopfGetValue Method
sopfSetValue Method
sopfTDequeLinkableInitDDM Method
sopfTDequeLinkableInitDD Method

Overriding Methods

sopfDefaultInit Method

somfGetValue Method

Gets the value from a **somf_TDequeLinkable** node.

IDL Syntax

```
somf_MCollectible somfGetValue ( );
```

Description

The **somfGetValue** method gets the value of the **somf_TDequeLinkable** object (node) represented by the receiving object. The method returns a pointer to a **somf_MCollectible Class** object containing the value.

Parameters

receiver

A pointer to an object of class **somf_TDequeLinkable**.

ev

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

Return Value

This method returns a pointer to a **somf_MCollectible** object containing the value obtained from the **somf_TDequeLinkable** object.

Example

```
somf_TDequeLinkable dl;
<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();
dl = somf_TDequeLinkableNew();

_somfSetValue(dl, ev, obj);
obj2 = (<Your Class which inherits from somf_MCollectible>*)
    _somfGetValue(dl, ev);

_somFree (dl);
_somFree (obj);
```

Original Class

somf_TDequeLinkable

Related Information

somfSetValue Method

somfSetValue Method

Sets the v alue of a given **somf_TDequeLinkable** node.

IDL Syntax

```
void somfSetValue (in somf_MCollectible v);
```

Description

The **somfSetValue** method sets the value of the **somf_TDequeLinkable** object (node) represented by the receiving object.

Parameters

receiver

A pointer to an object of class **somf_TDequeLinkable**.

ev

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

v

A pointer to the new value of the **somf_TDequeLinkable** object.

Example

```
somf_TDequeLinkable dl;
<your Class which inherits from somf_MCollectible> obj;
Environment *ev;

ev = somGetGlobalEnvironment();

obj = <your Class which inherits from somf_MCollectible>New();
dl = somf_TDequeLinkableNew();

_somfSetValue(dl, ev, obj);

_somFree (dl);
_somFree (obj);
```

Original Class

somf_TDequeLinkable

Related Information

somfGetValue Method

somfTDequeLinkableInitDD Method

Initializes a new **somf_TDequeLinkable** node, by specifying the adjacent nodes to which it will link. This method does not set a value for the node.

IDL Syntax

```
somf_TDequeLinkable somfTDequeLinkableInitDD (
    in somf_TDequeLinkable previous,
    in somf_TDequeLinkable next);
```

Description

The **somfTDequeLinkableInitDD** method initializes a new object (node) of class **somf_TDequeLinkable**. The method specifies the previous and next nodes to which the new node will link. However, it does not set a value for the node.

Note: You cannot override this method.

Parameters

receiver

A pointer to an object of class **somf_TDequeLinkable**.

ev

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

previous

A pointer to the **somf_TDequeLinkable** before this one.

next

A pointer to the **somf_TDequeLinkable** after this one.

Return Value

This method returns a pointer to the initialized **somf_TDequeLinkable** object that represents a node in a deque collection.

Example

```
somf_TDequeLinkable dll;
Environment *ev;

ev = somGetGlobalEnvironment();

dll = somf_TDequeLinkableNew();
_somfTDequeLinkableInitDD(dll, ev, SOMF_NIL, SOMF_NIL);

_somFree (dll);
```

Original Class

somf_TDequeLinkable

Related Information

somfTDequeLinkableInitDDM Method

somfTDequeLinkableInitDDM Method

initializes a new **somf_TDequeLinkable** node. This includes specifying the adjacent nodes and setting the value of the node.

IDL Syntax

```
somf_TDequeLinkable somfTDequeLinkableInitDDM(
    in somf_TDequeLinkable previous,
    in somf_TDequeLinkable next,
    in somf_MCollectible value);
```

Description

The **somfTDequeLinkableInitDDM** method initializes a new object (node) of class **somf_TDequeLinkable**. The method specifies the previous and next nodes to which the new node will link, and it also passes a value for the new node.

Note: You cannot override this method.

Parameters

receiver

A pointer to an object of class **somf_TDequeLinkable**.

ev

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

previous

A pointer to the **somf_TDequeLinkable** before this one.

value

A pointer to the value of this **somf_TDequeLinkable** object that represents a node in a deque collection.

Return Value

This method returns a pointer to the initialized **somf_TDequeLinkable** object (node).

Example

```
somf_TDequeLinkable dl2;
Environment *ev;

ev = somGetGlobalEnvironment();

dl2 = somf_TDequeLinkableNew();
_somfTDequeLinkableInitDDM(dl2, ev, SOMF_NIL, SOMF_NIL, SOMF_NIL);

_somFree (dl2);
```

Original Class

somf_TDequeLinkable

Related Information

somfTDequeLinkableInitDD Method

somfTDequeLinkableInitDDM Method

sopf_TDictionary Class

This class represents a collection of (*key*, *value*) pairs. 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 **sopf_THashTable Class**, each entry in a **sopf_TDictionary** is actually an object of the **sopf_TAssoc Class** that holds a pair. In most cases, this use of a **sopf_TAssoc** object is transparent to the user. However, you need to be aware of this **sopf_TAssoc** usage, because some **sopf_TDictionary** methods address the data as separate *key* and *value* parts of a pair, whereas other methods accept or return a single **sopf_TAssoc** object representing the pair.

The **sopf_TDictionary** class is very similar to **sopf_THashTable**. The primary difference is that **sopf_TDictionary** inherits from **sopf_TCollection Class**, whereas **sopf_THashTable** does not. The other distinction is that the **sopf_TDictionary** class uses the **sopfIsEqual Method** as its default comparison function, whereas **sopf_THashTable** uses **sopfIsSame Method**. The **sopf_TDictionary** class's use of **sopfIsEqual** means that equal keys can only appear in the dictionary once.

Objects inserted into a **sopf_TDictionary** collection must inherit from class **sopf_MCollectible Class**. In addition, they must override the **sopfHash Method** and the **sopfIsEqual** method. These methods are used internally by collections of the **sopf_TDictionary** class.

Because **sopf_TDictionary** takes **sopf_MCollectible** objects as members, any class that inherits from **sopf_MCollectible** can be a member of the dictionary. This means that you can have a dictionary containing objects of any main collection class.

Note: The **sopf_TDictionary** class uses the **sopfIsEqual** 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 **sopfIsEqual** method to equate entries, use the initialization methods to change to the **sopfIsSame** method.

Note: The **sopf_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: **sopmk**

Although 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

sopf_TCollection Class

Metaclass

SOMClass

Ancestor Classes

- somf_TCollection Class
- somf_MCollectible Class
- SOMObject Class

New Methods

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

Overriding Methods

- somDefaultInit Method
- somDestruct Method
- somfAdd Method
- somfCount Method
- somfCreateIterator Method
- somfDeleteAll Method
- somfMember Method
- somfRemove Method
- somfRemoveAll Method

somfAdd Method

Adds a specified *obj* 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 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 pairs of the **somf_TAssoc Class**. You cannot use this interface to add a generic **somf_MCollectible Class** 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** 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.

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 Class (overridden here)

Related Information

somfAddKeyValuePairMM Method
somfAddKeyValuePairMMB Method

sopfAddKeyValuePairMM Method

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

IDL Syntax

```
sopf_MCollectible sopfAddKeyValuePairMM (
    in sopf_MCollectible key,
    in sopf_MCollectible val);
```

Description

The **sopfAddKeyValuePairMM** method adds the specified pair to the dictionary, even if there is an existing 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 **sopf_TAssoc Class**, before adding the new pair to the dictionary.

Parameters

receiver

A pointer to an object of class **sopf_TDictionary**.

ev

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

key

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

val

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

Return Value

- **sopf_MCollectible**, a pointer to the **sopf_MCollectible** value that had to be removed in order for a new pair to be added.
- **SOPF_NIL**, no **sopf_MCollectible** value had to be removed in order to add the pair.

Example

```
sopf_TDictionary d;
<Your Class which inherits from sopf_MCollectible> obj;
<Your Class which inherits from sopf_MCollectible> obj2;
Environment *ev;

ev = sopfGetGlobalEnvironment();

obj = <Your Class which inherits from sopf_MCollectible>New();
obj2 = <Your Class which inherits from sopf_MCollectible>New();
d = sopf_TDictionaryNew();

if (_sopfAddKeyValuePairMM(d, ev, obj, obj2) != SOPF_NIL)
    sopfPrintf("\n problem adding obj,obj2 to d\n");

_sopfFree (d);
_sopfFree (obj);
_sopfFree (obj2);
```

Original Class

somf_TDictionary

Related Information

somfAdd Method

somfAddKeyValuePairMMB Method

somfAddKeyValuePairMMB Method

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 pair to the dictionary represented by the receiving object, unless the boolean argument *replace* prohibits this replacement.

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

Using the specified *key* and *val* arguments, this method transparently creates an object of the **somf_TAssoc Class**, before adding the new 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 if an existing pair with an identical key should be replaced.

Return Value

- **somf_MCollectible**, a pointer to the **somf_MCollectible** value that had to be removed in order for a new 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

somfAdd Method

somfAddKeyValuePairMM Method

somfAssign Method

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. The method sets/resets the instance variables of the receiver to the values of the source. This operation is logically equivalent to using the equal (=) 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. 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.

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

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

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. 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 Class (overridden here)

somfCreateliterator Method

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

IDL Syntax

```
somf_Titerator somfCreateliterator ( );
```

Description

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

This is one of two ways to initialize a **somf_TDictionaryIterator Class** to point to an instance of a **somf_TDictionary**. The other way is to use the **somf_TDictionaryIterator's** initializer 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 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 Class (overridden here)

somfCreateNewImplementationF Method

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 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 **somflsEqual** or a **somflsSame** method.

This argument should always be set to either

```
somf_MCollectibleClassData.somflsSame or  
somf_MCollectibleClassData.somflsEqual
```

This specification is necessary because SOM needs a pointer to the original declaration of the method, which resides in **somf_MCollectible Class**. The **somf_TDictionary** object will use this pointer to access the **somflsSame Method** or **somflsEqual 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.somflsEqual);  
  
_somFree (d);  
_somFree (ht2);
```

Original Class

somf_TDictionary

Related Information

somfCreateNewImplementationFL Method
somfCreateNewImplementationFLL Method
somfCreateNewImplementationFLLL Method

somfCreateNewImplementationFL Method

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 **somflsEqual** or a **somflsSame** method.

This argument should always be set to either

```
somf_MCollectibleClassData.somflsSame or  
somf_MCollectibleClassData.somflsEqual
```

because SOM needs a pointer to the original declaration of the method, which resides in **somf_MCollectible Class**. The **somf_TDictionary** object will use this pointer to access the **somflsSame Method** or **somflsEqual 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 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

somfCreateNewImplementationF Method
somfCreateNewImplementationFLL Method
somfCreateNewImplementationFLLL Method

somfCreateNewImplementationFLL Method

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 **somflsEqual** or a **somflsSame** method.

This argument should always be set to either

```
somf_MCollectibleClassData.somfIsSame or  

somf_MCollectibleClassData.somfIsEqual
```

because SOM needs a pointer to the original declaration of the method, which resides in **somf_MCollectible Class**. The **somf_TDictionary** object will use this pointer to access the **somflsSame Method** or **somflsEqual 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 pairs to expect.

rate

The growth rate of the hash table in the dictionary, expressed as the number of 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

somfCreateNewImplementationF Method
somfCreateNewImplementationFL Method
somfCreateNewImplementationFLLL Method

somfCreateNewImplementationFLLL Method

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 **somflsEqual** or a **somflsSame** method.

This argument should always be set to either

```
somf_MCollectibleClassData.somflsSame or  

somf_MCollectibleClassData.somflsEqual
```

because SOM needs a pointer to the original declaration of the method, which resides in **somf_MCollectible Class**. The **somf_TDictionary** object will use this pointer to access the **somflsSame Method** or **somflsEqual 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 pairs to expect.

rate

The growth rate of the hash table in the dictionary, expressed as the number of 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

somfCreateNewImplementationF Method
somfCreateNewImplementationFL Method
somfCreateNewImplementationFLL Method

somfDeleteAll Method

Removes all of the pairs from a dictionary and deallocates the storage that these objects might have owned. 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.

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 Method** 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. 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.

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 Class (overridden here)

Related Information

somfDeleteAllKeys Method
somfDeleteAllValues Method
somfDeleteKey Method

somfDeleteAllKeys Method

Removes all of the 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 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 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. 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.

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

somfDeleteAll Method
somfDeleteAllValues Method
somfDeleteKey Method

somfDeleteAllValues Method

Removes all of the 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 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 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 Method** 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. 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.

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

somfDeleteAll Method
somfDeleteAllKeys Method
somfDeleteKey Method

somfDeleteKey Method

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

IDL Syntax

```
somf_MCollectible somfDeleteKey (in somf_MCollectible key);
```

Description

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

- **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

somfDeleteAll Method
somfDeleteAllKeys Method
somfDeleteAllValues Method

somfDeleteKey Method

somfGetHashFunction Method

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. 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

somfSetHashFunction Method

somfKeyAtM Method

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. 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 Class** that is the value to be searched for.

Return Value

- **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

somfKeyAtMF Method

somfKeyAtMF Method

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 **somflsEqual** or **somflsSame** should be used to compare the values.

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 Class** object that is the value to be searched for.

testfn

A method pointer specifying either a **somflsEqual** or a **somflsSame** method.

This argument should always be set to either:

```
somf_MCollectibleClassData.somflsSame or
somf_MCollectibleClassData.somflsEqual.
```

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 **somflsSame** or **somflsEqual** method that was declared and defined in the object being inserted into, or removed from, the **somf_TDictionary** object.

Return Value

- **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.somflsEqual)
```

somfKeyAtMF Method

```
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

somfKeyAtM Method

somfMember Method

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

IDL Syntax

```
somf_MCollectible somfMember (in somf_MCollectible obj);
```

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. 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 Class** key of the (key, value) pair that may or may not be in the dictionary.

Return Value

- **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.

```
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 Class (overridden here)

somfRemove Method

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. 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 Class** object representing the key to be removed from the dictionary.

Return Value

- **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 Class (overridden here)

Related Information

somfRemoveAll Method

somfRemoveAll Method

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. 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.

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 Class (overridden here)

Related Information

somfRemove Method

somfSetHashFunction Method

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. 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 Class**. 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.

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

somfGetHashFunction Method

somfTDictionaryInitD Method

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 the receiving object will be equal to.

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

somfTDictionaryInitF Method
somfTDictionaryInitFL Method
somfTDictionaryInitFLL Method
somfTDictionaryInitL Method
somfTDictionaryInitLL Method
somfTDictionaryInitLLF Method

somfTDictionaryInitF Method

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 **somflsEqual** or a **somflsSame** method.

This argument should always be set to either

```
somf_MCollectibleClassData.somfIsSame or
somf_MCollectibleClassData.somfIsEqual
```

because SOM needs a pointer to the original declaration of the method, which resides in **somf_MCollectible Class**. The **somf_TDictionary** object will use this pointer to access the **somflsSame Method** or **somflsEqual Method** that was declared and defined in the inserted or removed object.

Return Value

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

somfTDictionaryInitFL Method

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 **somflsEqual** or a **somflsSame** method.

This argument should always be set to either

```
somf_MCollectibleClassData.somfIsSame or  

somf_MCollectibleClassData.somfIsEqual.
```

because SOM needs a pointer to the original declaration of the method, which resides in **somf_MCollectible Class**. The **somf_TDictionary** object will use this pointer to access the **somflsSame Method** or **somflsEqual Method** that was declared and defined in the inserted or removed 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

somfTDictionaryInitFL Method

Related Information

somfTDictionaryInitD Method
somfTDictionaryInitF Method
somfTDictionaryInitFLL Method
somfTDictionaryInitL Method
somfTDictionaryInitLL Method
somfTDictionaryInitLLF Method

somfTDictionaryInitFLL Method

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 **somflsEqual** or a **somflsSame** method.

This argument should always be set to either

```
somf_MCollectibleClassData.somflsSame or
somf_MCollectibleClassData.somflsEqual.
```

because SOM needs a pointer to the original declaration of the method, which resides in **somf_MCollectible Class**. The **somf_TDictionary** object will use this pointer to access the **somflsSame Method** or **somflsEqual Method** that was declared and defined in the inserted or removed 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 Method

```
_somfTDictionaryInitFLL(d1, ev,  
    somf_MCollectibleClassData.somfIsEqual, 8, 8);  
  
_somFree (d1);
```

Original Class

somf_TDictionary

Related Information

somfTDictionaryInitD Method
somfTDictionaryInitF Method
somfTDictionaryInitFL Method
somfTDictionaryInitL Method
somfTDictionaryInitLL Method
somfTDictionaryInitLLF Method

somfTDictionaryInitL Method

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 **somflsEqual Method** is used unless the **somfSetHashFunction Method** has changed it to **somflsSame 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.

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

somfTDictionaryInitD Method
somfTDictionaryInitF Method
somfTDictionaryInitFL Method
somfTDictionaryInitFLL Method
somfTDictionaryInitLL Method
somfTDictionaryInitLLF Method

somfTDictionaryInitLL Method

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 **somflsEqual Method** is used as the comparison method, unless the **somfSetHashFunction Method** has changed it to **somflsSame 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.

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

somfTDictionaryInitD Method
somfTDictionaryInitF Method
somfTDictionaryInitFL Method
somfTDictionaryInitFLL Method
somfTDictionaryInitL Method
somfTDictionaryInitLLF Method

somfTDictionaryInitLLF Method

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 **somflsEqual** or **somflsSame** method.

This argument should always be set to either

```
somf_MCollectibleClassData.somflsSame or
somf_MCollectibleClassData.somflsEqual.
```

because SOM needs a pointer to the original declaration of the method, which resides in **somf_MCollectible Class**. The **somf_TDictionary** object will use this pointer to access the **somflsSame Method** or **somflsEqual Method** that was declared and defined in the inserted or removed 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.somflsEqual);

_somFree (d4);
```

somfTDictionaryInitLLF Method

Original Class

somf_TDictionary

Related Information

somfTDictionaryInitD Method
somfTDictionaryInitF Method
somfTDictionaryInitFL Method
somfTDictionaryInitFLL Method
somfTDictionaryInitL Method
somfTDictionaryInitLL Method

somfValueAt Method

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 Class** object that is the key to be searched for.

Return Value

- **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

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**

Note: 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 Class

Metaclass

SOMClass

Ancestor Classes

somf_TIterator Class
SOMObject Class

New Methods

somfTDictionaryIteratorInit Method

Overriding Methods

somDefaultInit Method
somDestruct Method
somfFirst Method
somfNext Method
somfRemove Method

somfFirst Method

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 Class** 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 (example: **somf_TSequence**, **somf_TIterator**). 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

- A pointer to the first **somf_MCollectible** object in the dictionary.
- 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)
{
```

somfFirst Method

```
    objk = _somfGetKey(itrojb, ev);
    objv = _somfGetValue(itrojb, ev);

    /* Do something with objk or objv */

    itrojb = _somfNext(itr, ev);
}

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

Original Class

somf_TIterator Class (overridden here)

Related Information

somfNext Method

somfNext Method

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 Class** object.

If the dictionary has changed since the last time **somfFirst** was called (other than through the use of the **somfRemove Method** 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 (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

- **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)
```

somfNext Method

```
{
    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_Tliterator Class (overridden here)

Related Information

somfFirst Method

somfRemove Method

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 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. You will probably have to fully qualify the method name. 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.

Example

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

ev = somGetGlobalEnvironment();

d = somfdictionaryinit();
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 Class (overridden here)

somfTDictionaryIteratorInit Method

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 the **somf_TDictionaryIterator** class, given the **somf_TDictionary Class** dictionary over which iteration is needed.

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**.

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

sopf_THashTable Class

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 and 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: **sopmk**

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

The **sopf_THashTable** class is very similar to **sopf_TDictionary**. The primary difference is that **sopf_THashTable** inherits directly from the **sopf_MCollectible Class**, whereas **sopf_TDictionary** is another level down, inheriting from **sopf_TCollection Class**. The other distinction is that the **sopf_THashTable** class uses the **sopfIlsSame Method** as its default comparison function, whereas **sopf_TDictionary** uses **sopfIlsEqual Method**.

Objects inserted into a **sopf_THashTable** collection must inherit from **sopf_MCollectible**. In addition, they should override the **sopfHash Method**, and the **sopfIlsEqual** method. These methods are used internally by objects of the **sopf_THashTable** class.

Because **sopf_THashTable** takes **sopf_MCollectible** objects as members, any class that inherits from **sopf_MCollectible** can be a member of the hash table. This means you can have a hash table containing objects of any main collection class.

Note: The **sopf_THashTable** class uses the **sopfIlsSame** 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 do not want to use the **sopfIlsSame** method to equate entries, use one of the initialization methods to change to the **sopfIlsEqual** method. Just be aware that if the comparison methods are changed, the objects inserted into the **sopf_THashTable** must have **sopfIlsEqual** and **sopfHash** overridden.

Note: This Hash Table does not allow two 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

sopf_MCollectible Class

Metaclass

SOMClass

Ancestor Classes

somf_MCollectible Class
SOMObject

New Methods

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

Overriding Methods

somDefaultInit Method
somDestruct Method

somfAddMM Method

Adds a pair to the hash table. This method will replace a copy if one already exists.

IDL Syntax

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

Description

The **somfAddMM** method adds the specified pair to the hash table. If it contains an existing pair for *key*; the method removes the existing value, adds the new *value* and returns the existing value of the conflicting pair. Using the arguments, this method transparently creates an object of the **somf_TAssoc Class**, before adding the new 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

- **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 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

somfAddMMB Method

somfAddMMB Method

Adds a 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 pair to the hash table represented by the receiving object, unless the boolean argument prohibits replacement of a conflicting pair.

If `replace = TRUE`, the pair is added to the hash table regardless of whether a copy (a pair having the same key) already exists. Otherwise, if `replace = FALSE`, then the 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 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

- **somf_MCollectible**, a pointer to **thesomf_MCollectible** value that had to be removed in order to add a new 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****somfAddMM Method**

somfAssign Method

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. The method sets/resets the instance variables of the receiver to the values of the source. This operation is logically equivalent to using the equal (=) 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. 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.

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

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. 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

sopfDelete Method

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

IDL Syntax

```
sopf_MCollectible sopfDelete (in sopf_MCollectible key);
```

Description

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

A hash table does not contain copies of the objects, but pointers to the objects. Using **sopfDelete** deletes the original object. If an attempt is made to delete a **sopf_MCollectible Class** object already deleted, this will cause a segmentation violation.

Note: You cannot override this method.

Parameters

receiver

A pointer to an object of class **sopf_THashTable**.

ev

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

key

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

Return Value

- **sopf_MCollectible**, a pointer to the value that was removed because the key was deleted.
- **SOMF_NIL**, the key object was not found.

Example

```
sopf_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 (_sopfDelete(ht, ev, obj) == SOMF_NIL)
    somPrintf(" Why did Delete say obj wasn't in ht? \n");

_sopfFree (ht);
```

Original Class

sopf_THashTable

Related Information

sopfDeleteAll Method

sopfDeleteAllKeys Method

sopfDeleteAllValues Method

somfDeleteAll Method

Removes all of the pairs from a hash table and deallocates the storage that these pairs might have owned.

IDL Syntax

```
void somfDeleteAll ( );
```

Description

The **somfDeleteAll** method removes all of the 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 Method** 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.

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

somfDelete Method

somfDeleteAllKeys Method

somfDeleteAllValues Method

somfDeleteAllKeys Method

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

IDL Syntax

```
void somfDeleteAllKeys ( );
```

Description

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

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. If an attempt is made to delete a **somf_MCollectible Class** 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. 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.

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

somfDeleteAll Method

somfDelete Method

somfDeleteAllValues Method

somfDeleteAllValues Method

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

IDL Syntax

```
void somfDeleteAllValues ( );
```

Description

The **somfDeleteAllValues** method removes all of the 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. The destructor function is called for each value in the hash table. However, the program still owns the objects representing the keys of the pairs.

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. If an attempt is made to delete a **somf_MCollectible Class** 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. 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.

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

somfDeleteAll Method
somfDeleteAllKeys Method
somfDelete Method

somfGetGrowthRate Method

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

somfSetGrowthRate Method

somfGetHashFunction Method

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. 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

somfSetHashFunction Method

somfGetRehashThreshold Method

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

somfSetRehashThreshold Method

somfGrow Method

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.

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

sopfMember Method

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

IDL Syntax

```
sopf_MCollectible sopfMember (in sopf_MCollectible key);
```

Description

The **sopfMember** method determines whether the 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 **sopf_TCollection** is used with a child of **sopf_THashTable**, then the name of the method will have to be fully qualified. 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->sopfMember (ev, key);
```

Note: You cannot override this method.

Parameters

receiver

A pointer to an object of class **sopf_THashTable**.

ev

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

key

A pointer to the **sopf_MCollectible** key of the pair that may or may not be a member of the Hash Table.

Return Value

- **sopf_MCollectible**, a pointer to the key of the pair the method determined as the member.
- **SOPF_NIL**, the object was not found.

Example

```
sopf_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 (sopf_THashTable_sopfMember(ht, ev,obj) != SOPF_NIL)
    somPrintf("\n obj IS in ht\n");
else
    somPrintf("\n obj is NOT in ht\n");
_sopfFree (ht);
```

Original Class

sopf_THashTable

sopfRemove Method

Removes from the hash table the pair associated with a given key.

IDL Syntax

```
sopf_MCollectible sopfRemove (in sopf_MCollectible key);
```

Description

The **sopfRemove** method removes from the hash table the 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. **sopfRemove** is a method name declared in multiple parents. You will probably have to fully qualify the method name. 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->sopfremoveall(ev, key);
```

Note: You cannot override this method.

Parameters

receiver

A pointer to an object of class **sopf_THashTable**.

ev

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

key

A pointer to the **sopf_MCollectible** key of the pair to be removed.

Return Value

- **sopf_MCollectible**. a pointer to the value of the pair that was removed.
- **SOPF_NIL**, the *key* object was not found.

Example

```
sopf_THashTable ht;
<your Class which inherits from sopf_MCollectible> key;
Environment *ev;

ev = sopfGetGlobalEnvironment();

ht = sopf_THashTableNew();
key = <your Class which inherits from sopf_MCollectible>New();

/* Add a lot of objects to ht */

if (sopf_THashTable_sopfRemove(ht, ev, key) == SOPF_NIL)
    sopfPrintf(" Why did Remove say key was not removed?\n");

_sopfFree (ht);
_sopfFree (key);
```

Original Class

sopf_THashTable

Related Information

sopfRemoveAll Method

somfRemoveAll Method

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

IDL Syntax

```
void somfRemoveAll ( );
```

Description

The **somfRemoveAll** method removes all of the 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. 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.

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

somfRemove Method

somfRetrieve Method

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 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

- **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

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 pairs by which to expand the table size when it grows.

Example

```
somf_THashTable ht;
Environment *ev;

ev = somGetGlobalEnvironment();

ht = somf_THashTableNew();

_somfSetGrowthRate(ht, ev, 20);

_somFree (ht);
```

Original Class

somf_THashTable

Related Information

somfGetGrowthRate Method

somfSetHashFunction Method

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
```

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 inserted or removed object.

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

somfGetHashFunction Method

somfSetRehashThreshold Method

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%.

Example

```
somf_THashTable ht;
Environment *ev;

ev = somGetGlobalEnvironment();

ht = somf_THashTableNew();

_somfSetRehashThreshold(ht, ev, 80);

_somFree (ht);
```

Original Class

somf_THashTable

Related Information

somfGetRehashThreshold Method

somfTHashTableInitFL Method

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 **somflsEqual** or a **somflsSame** method. This method is used to compare two keys in the hash table.

This argument should always be set to either

```
somf_MCollectibleClassData.somflsSame or
somf_MCollectibleClassData.somflsEqual.
```

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 **somflsSame Method** or **somflsEqual Method** that was declared and defined in the inserted or removed object.

tablesize

The initial size of the hash table, expressed as the number of 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.somflsEqual, 23);
_somFree (h3);
```

Original Class

somf_THashTable

Related Information

somfTHashTableInitFLLL Method
somfTHashTableInitFLL Method
somfTHashTableInitH Method

somfTHashTableInitFLL Method

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 **somflsEqual** or a **somflsSame** method. This method is used to compare two keys in the hash table.

This argument should always be set to either

```
somf_MCollectibleClassData.somflsSame or
somf_MCollectibleClassData.somflsEqual.
```

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 **somflsSame Method** or **somflsEqual 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 pairs that are expected.

rate

The growth rate, expressed as the number of 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.somflsEqual, 23, 20);

_somFree (h2);
```

Original Class

`somf_THashTable`

Related Information

`somfTHashTableInitFLLL Method`

`somfTHashTableInitFL Method`

`somfTHashTableInitH Method`

somfTHashTableInitFLLL Method

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 **somflsEqual** or a **somflsSame** method. This method is used to compare two keys in the hash table.

This argument should always be set to either

```
somf_MCollectibleClassData.somflsSame or
somf_MCollectibleClassData.somflsEqual.
```

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 **somflsSame Method** or **somflsEqual 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 pairs that are expected.

rate

The growth rate, expressed as the number of 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

somfTHashTableInitFLL Method

somfTHashTableInitFL Method

somfTHashTableInitH Method

somfTHashTableInitH Method

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

somfTHashTableInitFLLL Method

somfTHashTableInitFLL Method

somfTHashTableInitFL Method

somf_THashTableIterator Class

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**

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 Class

Metaclass

SOMClass

Ancestor Classes

somf_TIterator Class
SOMObject

New Methods

somfTHashTableIteratorInit

Overriding Methods

somDefaultInit Method
somDestruct Method
somfFirst Method
somfNext Method
somfRemove Method

somfFirst Method

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. You will probably have to fully qualify the method name. 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 Class (overridden here)

Related Information

somfNext Method

somfNext Method

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 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.

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 Class** 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. If **somfAdd** 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. 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. 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

- **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_Tlterator Class (overridden here)

Related Information

somfFirst Method

somfRemove Method

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 Method** or **somfNext Method**) 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. You will probably have to fully qualify the method name. 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.

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 Class (overridden here)

somfTHashTableIteratorInit Method

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.

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

sopf_Tlterator Class

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 **sopf_Tlterator** 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: **sopmk**

If you create classes that inherit from the **sopf_Tlterator** class, the new classes must override the methods **sopfFirst** and **sopfNext**.

When creating an iterator for an unordered collection, your classes should inherit from **sopf_Tlterator**. (When creating an iterator for an ordered collection, your classes should inherit from **sopf_TSequenceliterator Class**). The **sopf_Tlterator** 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

sopfNext Method
sopfFirst Method
sopfRemove Method

Overriding Methods

None

somfFirst Method

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 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. **somfFirst** is a method name declared in multiple parents (example: **somf_TSequence**, **somf_TIterator**, etc.). You will probably have to fully qualify the method name (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 Class** or **somf_TDictionaryIterator Class**, or any of the other classes that inherit from **somf_TIterator**.

Original Class

somf_TIterator

Related Information

somfNext Method

somfNext Method

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 Method** 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

- **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 Class** or **somf_TDictionaryIterator Class**, or any of the other classes that inherit from **somf_TIterator**.

Original Class

somf_TIterator

Related Information

somfFirst Method

somfRemove Method

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 Method** or **somfNext Method**) 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 (example: **somf_TCollection**, **somf_THashTable**, **somf_TIterator**, etc.) You will probably have to fully qualify the method name. 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.

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 Class** or **somf_TDictionaryIterator Class**, or any of the other classes that inherit from **somf_TIterator**.

Original Class

somf_TIterator

somf_TPrimitiveLinkedList Class

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

Note: 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 Method
somfRemove Method
somfRemoveAll Method
somfRemoveFirst Method
somfLast Method
somfAddBefore Method
somfAddAfter Method
somfAddFirst Method
somfAddLast Method
somfAfter Method
somfBefore Method
somfFirst Method
somfLast Method

Overriding Methods

somDefaultInit Method
somDestruct Method

somfAddAfter Method

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

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.

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

somfAddBefore Method

somfAddFirst Method

somfAddLast Method

somfAddBefore Method

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

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.

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

somfAddAfter Method
somfAddFirst Method
somfAddLast Method

somfAddFirst Method

Adds an object as the first object in a list.

IDL Syntax

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

Description

The **somfAddFirst** method adds 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.

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

somfAddAfter Method
somfAddBefore Method
somfAddLast Method

somfAddLast Method

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.

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

somfAddAfter Method
somfAddBefore Method
somfAddFirst Method

somfAfter Method

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 *obj*.

Return Value

- **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

somfBefore Method

somfBefore Method

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 *obj*.

Return Value

- **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

somfAfter Method

somfCount Method

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

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

- **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

somfLast Method

somfLast Method

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_TSequenceliterator**, **somf_TSequence**). You will probably have to fully qualify the name of the method. 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

- **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

somfFirst Method

somfRemove Method

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. You will probably have to fully qualify the name of the method. 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:

```
p11->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.

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

somfRemoveAll Method

somfRemoveFirst Method

somfRemoveLast Method

somfRemoveAll Method

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.

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

somfRemove Method

somfRemoveFirst Method

somfRemoveLast Method

somfRemoveFirst Method

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

- **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

somfRemove Method

somfRemoveAll Method

somfRemoveLast Method

sopfRemoveLast Method

Removes the last object from a given list.

IDL Syntax

```
sopf_MLinkable sopfRemoveLast ( );
```

Description

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

Parameters

receiver

A pointer to an object of class **sopf_TPrimitiveLinkedList**.

ev

A pointer to the Environment structure for the calling method.

Return Value

- **sopf_MLinkable**, a pointer to the **sopf_MLinkable** object removed from the list.
- **SOPF_NIL**, nothing is in the list.

Example

```
sopf_TPrimitiveLinkedList l;
Environment *ev;

ev = somGetGlobalEnvironment();

l = somf_TPrimitiveLinkedListNew();

/* Add some objects to l */

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

_sopfFree (l);
```

Original Class

sopf_TPrimitiveLinkedList

Related Information

sopfRemove Method

sopfRemoveAll Method

sopfRemoveFirst Method

somf_TPrimitiveLinkedListIterator Class

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 Method
somfNext Method
somfLast Method
somfPrevious Method
somfTPrimitiveLinkedListIteratorInit Method

Overriding Methods

somDestruct Method

somfFirst Method

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. The **somf_TPrimitiveLinkedListIterator** class does not inherit from **somf_TIterator Class**. 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. You will probably have to fully qualify the name of the method. 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

- **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

somfNext Method

somfLast Method

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. You will probably have to fully qualify the name of the method. 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

- **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

somfPrevious Method

somfNext Method

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. The **somf_TPrimitiveLinkedListIterator** class does not inherit from **somf_TIterator Class**.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. If any child of **somf_TPrimitiveLinkedListIterator** is used with **somf_TIterator**, then the name of the method will have to be fully qualified. 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

- **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

somfFirst Method

sopfPrevious Method

Gets the previous object from a given list.

IDL Syntax

```
sopf_MLinkable sopfPrevious ( );
```

Description

The **sopfPrevious** method determines the previous object in the list corresponding 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 **sopf_TPrimitiveLinkedListIterator** is used with **sopf_TSequenceliterator**, then the name of the method will have to be fully qualified (for example: **sopf_TPrimitiveLinkedListIterator_sopfPrevious**). 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->sopfPrevious(ev);
```

Parameters

receiver

A pointer to an object of class **sopf_TPrimitiveLinkedListIterator**.

ev

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

Return Value

- **sopf_MLinkable**, a pointer to the **sopf_MLinkable** object before the receiving object.
- **SOMF_NIL**, the beginning of the list has been reached.

Example

```
sopf_TPrimitiveLinkedList l;
sopf_MLinkable obj;
sopf_TPrimitiveLinkedListIterator itr;
Environment *ev;

ev = somGetGlobalEnvironment();

l = sopf_TPrimitiveLinkedListNew();
itr = sopf_TPrimitiveLinkedListIteratorNew();
_sopfTPrimitiveLinkedListIteratorInit(itr, ev, l);

/* Add a lot of objects to l */

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

_sopfFree (l);
_sopfFree (itr);
```

Original Class

sopf_TPrimitiveLinkedListIterator

Related Information

sopfLast Method

somfTPrimitiveLinkedListIteratorInit Method

Initializes a **somf_TPrimitiveLinkedListIterator** object, establishing it as the iterator for a given **somf_TPrimitiveLinkedList Class** 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

receive

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

sopf_TPriorityQueue Class

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

Robert Sedgewick in *Algorithms in C++* 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. 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 and stacks is called a priority queue.

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

Note: The **sopf_TPriorityQueue** class uses the **sopfIsEqual** 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 **sopfIsEqual** method to equate entries, use the initialization methods to change to the **sopfIsSame** method.

Objects that are inserted into a **sopf_TPriorityQueue** collection should override the methods **sopfIsEqual**, **sopfIsLessThan**, **sopfIsGreaterThan** and **sopfHash**.

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

sopf_TCollection Class

Metaclass

SOMClass

Ancestor Classes

sopf_TCollection Class
sopf_MCollectible Class
SOMObject

New Methods

sopfInsert Method
sopfPeek Method
sopfPop Method
sopfReplace Method
sopfSetEqualityComparisonFunction Method
sopfGetEqualityComparisonFunction Method
sopfAssign Method
sopfTPriorityQueueInitF Method
sopfTPriorityQueueInitP Method

Overriding Methods

- somDefaultInit Method**
- somDestruct Method**
- somfAdd Method**
- somfRemove Method**
- somfRemoveAll Method**
- somfDeleteAll Method**
- somfCount Method**
- somfMember Method**
- somfCreateIterator Method**

somfAdd Method

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 Class** 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 Class (overridden here)

Related Information

somfInsert Method

somfAssign Method

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 equal (=) 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.

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

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 Class (overridden here)

somfCreateliterator Method

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

IDL Syntax

```
somf_Titerator somfCreateliterator ( );
```

Description

The **somfCreateliterator** 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 Class** to point to an instance of the **somf_TPriorityQueue** class. The other way is to use the **somf_TPriorityQueueIterator**'s initializer method.

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 Class (overridden here)

somfDeleteAll Method

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 Method** 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.

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 Class (overridden here)

somfGetEqualityComparisonFunction Method

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

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 **somflsEqual** method.

Note: :Do not confuse this “equality compare function” with the **somfCompare Method** 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.somflsEqual)
{
    somPrintf("\n What Compare Function are we using?\n");
}

_somFree (pq);
```

Original Class

somf_TPriorityQueue

Related Information

somfSetEqualityComparisonFunction Method

somfInsert Method

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.

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

somfAdd Method

somfMember Method

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

- **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 Class (overridden here)

somfPeek Method

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

- **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

somfPop Method

somfPop Method

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

- **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

somfPeek Method

somfReplace Method

somfRemove Method

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. You will probably have to fully qualify the name of the method. 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

- **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 Class (overridden here)

Related Information

somfRemoveAll Method

somfRemoveAll Method

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.

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 Class (overridden here)

Related Information

somfRemove Method

somfReplace Method

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

- **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

somfPop Method
somfInsert Method

somfSetEqualityComparisonFunction Method

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 **somflsEqual**. Normally, this default function will not need to be changed.

Note: Do not confuse this “equality comparison function” with the **somfCompare Method** in **somf_MOrderableCollectible Class**. 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 **somflsEqual Method** or **somflsSame Method**.

This argument should always be set to either

```
somf_MCollectibleClassData.somflsSame or
somf_MCollectibleClassData.somflsEqual.
```

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

Example

```
somf_TPriorityQueue pq;
Environment *ev;

ev = somGetGlobalEnvironment();
pq = somf_TPriorityQueueNew();

/* Add some objects to pq */
_somfSetEqualityComparisonFunction(pq, ev,
    somf_MCollectibleClassData.somflsEqual);

_somFree (pq);
```

Original Class

somf_TPriorityQueue

Related Information

somfGetEqualityComparisonFunction Method

somfTPriorityQueueInitF Method

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. 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, **somflsLessThan** 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 **somflsLessThan Method** means that smaller objects are removed first and larger objects are removed last. Using the **somflsGreaterThan Method** reverses this.

This should always be set to either

```
somf_MOrderableCollectibleClassData.
    somflsLessThan or
somf_MOrderableCollectibleClassData.
    somflsGreaterThan
```

because SOM needs a pointer to the original declaration of the method, which resides in **somf_MOrderableCollectible Class**. The **somf_TPriorityQueue** object will use this pointer to access the **somflsLessThan** or **somflsGreaterThan** method that was declared and defined in the inserted or removed 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.somflsLessThan);
_somFree (pq1);
```

Original Class

somf_TPriorityQueue

Related Information

somfTPriorityQueueInitP Method

somfTPriorityQueueInitP Method

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

somfTPriorityQueueInitF Method

sopf_TPriorityQueueIterator Class

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

Note: A **sopf_TPriorityQueueIterator** iterator does not return objects in order, because a **sopf_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

sopf_TIterator Class

Metaclass

SOMClass

Ancestor Classes

sopf_TIterator Class
SOMClass

New Methods

sopfTPriorityQueueIteratorInit Method

Overriding Methods

sopfNext Method
sopfFirst Method
sopfRemove Method

somfFirst Method

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 returns the first object of the priority queue that corresponds to the specified iterator. This method resets the iterator to the beginning even if other operations on the collection cause the iterator to be invalidated.

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. You will probably have to fully qualify the method name. 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 Class (overridden here)

Related Information

somfNext Method

somfNext Method

Gets the next object in a priority queue.

IDL Syntax

```
somf_MCollectible somfNext ();
```

Description

somfNext gets the next object in the priority queue that corresponds to the iterator representing the receiving object and returns a pointer to it.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. 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. 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 **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

- **somf_MCollectible**, a pointer to the next **somf_MCollectible** in the 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 Class (overridden here)

Related Information

somfFirst Method

somfRemove Method

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 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 Method** 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. You will probably have to fully qualify the method name. 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.

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 Class

somfTPriorityQueueIteratorInit Method

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 iterator of class **somf_TPriorityQueueIterator**, given the **somf_TPriorityQueue Class** object over which iteration is needed. 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 is to use the **somf_TPriorityQueue** class's **somfCreateIterator Method**.

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

somf_TSequence Class

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

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

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

File Stem

tseq

Base

somf_TCollection Class

Metaclass

SOMClass

Ancestor Classes

somf_TCollection Class
somf_MCollectible Class
SOMObject

New Methods

somfFirst Method
somfAfter Method
somfBefore Method
somfLast Method
somfOccurrencesOf Method
somfTSequenceInit Method

Overriding Methods

somfAdd Method
somfRemove Method
somfRemoveAll Method
somfDeleteAll Method
somfCount Method
somfCreateIterator Method
somDefaultInit Method

sopfAdd Method

Adds an object to a given ordered collection.

IDL Syntax

```
sopf_MCollectible sopfAdd (in sopf_MCollectible obj);
```

Description

The **sopfAdd** method adds a specified object *obj* to the ordered collection represented by the receiving object. Every class that inherits from the **sopf_TSequence** class must override this method for that class to work correctly.

Parameters

receiver

A pointer to an object of class **sopf_TSequence**.

ev

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

obj

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

Return Value

- **sopf_MCollectible**, a pointer to the **sopf_MCollectible** object that had to be removed in order to add *obj*. Recall that some of the main collection classes will only accept one occurrence of an object where the **sopfIsEqual Method** or **sopfIsSame Method** would be TRUE.
- SOMF_NIL, no **sopf_MCollectible** object had to be removed in order to add *obj*.

Example

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

Original Class

sopf_TCollection Class (overridden here)

sopfAfter Method

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

IDL Syntax

```
sopf_MCollectible sopfAfter (in sopf_MCollectible obj);
```

Description

The **sopfAfter** method returns the object found after object *obj* in the ordered collection represented by the receiving object. Every class that inherits from the **sopf_TSequence** class must override this method for that class to work correctly.

Parameters

receiver

A pointer to an object of class **sopf_TSequence**.

ev

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

obj

A pointer to the **sopf_MCollectible** object that is in front of the returned obj.

Return Value

- **sopf_MCollectible**, a pointer to the **sopf_MCollectible** object after *obj*.
- **SOPF_NIL**, the *obj* is the last object in this collection or could not be found.

Example

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

Original Class

sopf_TSequence

Related Information

sopfBefore Method

sopfFirst Method

sopfLast Method

sopfBefore Method

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

IDL Syntax

```
sopf_MCollectible sopfBefore (in sopf_MCollectible obj);
```

Description

The **sopfBefore** method returns the object found immediately before the specified object *obj* in the ordered collection represented by the receiving object. Every class that inherits from the **sopf_TSequence** class must override this method for that class to work correctly.

Parameters

receiver

A pointer to an object of class **sopf_TSequence**.

ev

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

obj

A pointer to the **sopf_MCollectible** object that is behind the returned object.

Return Value

- **sopf_MCollectible**, a pointer to the **sopf_MCollectible** object that precedes *obj*.
- **SOPF_NIL**, the *obj* is the first object in this collection or could not be found.

Example

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

Original Class

sopf_TSequence

Related Information

sopfAfter Method
sopfFirst Method
sopfLast Method

somfCount Method

Gets the number of objects in this ordered collection.

IDL Syntax

```
long somfCount ( );
```

Description

The **somfCount** method determines the number of objects in the ordered collection represented by the receiving object, and returns that number. Every class that inherits from the **somf_TSequence** class must override this method for that class to work correctly.

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

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

Parameters

receiver

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

ev

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

Return Value

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

Example

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

Original Class

somf_TCollection Class (overridden here)

somfCreateliterator Method

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

IDL Syntax

```
somf_Titerator somfCreateliterator ( );
```

Description

The **somfCreateliterator** method returns a new iterator that is suitable for iterating over the objects in the ordered collection represented by the receiving object. Every class that inherits from the **somf_TSequence** class must override this method for that class to work correctly.

Parameters

receiver

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

ev

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

Return Value

This method returns a pointer to the new iterator.

Example

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

Original Class

somf_TCollection Class (overridden here)

somfDeleteAll Method

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

IDL Syntax

```
void somfDeleteAll ( );
```

Description

The **somfDeleteAll** method removes all of the objects from the ordered collection represented by the receiving object. The method also deallocates the storage that these objects might have owned (that is, the destructor function is called for each object in the collection). Every class that inherits from the **somf_TSequence** class must override this method for that class to work correctly.

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

ev

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

Example

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

Original Class

somf_TCollection Class (overridden here)

somfFirst Method

Gets the first object in an ordered collection.

IDL Syntax

```
somf_MCollectible somfFirst ( );
```

Description

The **somfFirst** method determines the first object in the ordered collection represented by the receiving object, and returns a pointer to it. Every class that inherits from the **somf_TSequence** class must override this method for that class to work correctly.

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

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

Parameters

receiver

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

ev

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

Return Value

- **somf_MCollectible**, a pointer to the first **somf_MCollectible** object in the ordered collection.
- **SOMF_NIL**, nothing is in the collection.

Example

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

Original Class

somf_TSequence

Related Information

somfLast Method

somfAfter Method

somfBefore Method

somfLast Method

Gets the last object in an ordered collection.

IDL Syntax

```
somf_MCollectible somfLast ( );
```

Description

The **somfLast** method determines the last object in the ordered collection represented by the receiving object, and returns a pointer to it. Every class that inherits from the **somf_TSequence** class must override this method for that class to work correctly.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. **somfLast** is a method name declared in multiple parents (example: **somf_TSequenceliterator**, **somf_TSequence**, etc.). You will probably have to fully qualify the method name (example: **somf_TDeque_somfLast**). This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

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

Parameters

receiver

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

ev

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

Return Value

- **somf_MCollectible**, a pointer to the last **somf_MCollectible** object in the ordered collection.
- **SOMF_NIL**, nothing is in the collection.

Example

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

Original Class

somf_TSequence

Related Information

somfAfter Method
somfBefore Method
somfFirst Method

somfOccurrencesOf Method

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

IDL Syntax

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

Description

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

Parameters

receiver

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

ev

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

obj

A pointer to the **somf_MCollectible** object to look for in the collection.

Return Value

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

Example

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

ev = somGetGlobalEnvironment();

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

_somfAddFirst(dq, ev, obj);

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

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

Original Class

somf_TSequence

somfRemove Method

Removes an object from an ordered collection.

IDL Syntax

```
somf_MCollectible somfRemove (in somf_MCollectible obj);
```

Description

The **somfRemove** method removes a specified object *obj* from the ordered collection represented by the receiving object. Every class that inherits from the **somf_TSequence Class** must override this method for that class to work correctly.

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

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

Parameters

receiver

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

ev

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

obj

A pointer to the **somf_MCollectible** object to be removed from the collection.

Return Value

- **somf_MCollectible**, a pointer to the object which was removed.
- **SOMF_NIL**, the object was not found.

Example

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

Original Class

somf_TCollection Class (overridden here)

Related Information

somfRemoveAll Method

somfRemoveAll Method

Removes all of the objects from an ordered collection.

IDL Syntax

```
void somfRemoveAll ( );
```

Description

The **somfRemoveAll** method removes all of the objects from the ordered collection represented by the receiving object. Every class that inherits from the **somf_TSequence** class must override this method for that class to work correctly.

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

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

Parameters

receiver

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

ev

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

Example

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

Original Class

somf_TCollection Class (overridden here)

Related Information

somfRemove Method

somfTSequenceInit Method

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

IDL Syntax

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

Description

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

Note: You cannot override this method.

Parameters

receiver

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

ev

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

testfn

A method pointer specifying either a **somflsEqual** or a *somflsSame* method.

This argument should always be set to either

```
somf_MCollectibleClassData.somflsIsSame or  
somf_MCollectibleClassData.somflsIsEqual.
```

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

Return Value

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

Original Class

somf_TSequence

somf_TSequenceliterator Class

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

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

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

File Stem

tseqitr

Base

somf_Titerator Class

Metaclass

SOMClass

Ancestor Classes

somf_Titerator Class
SOMObject

New Methods

somfLast Method
somfPrevious Method

Overriding Methods

somfFirst Method
somfNext Method
somfRemove Method

somfFirst Method

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

IDL Syntax

```
somf_MCollectible somfFirst ( );
```

Description

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

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

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

C cannot handle methods from different classes having the same name when they inherit the name from different parents. **somfFirst** is a method name declared in multiple parents (example: **somf_TSequence**, **somf_TIterator**, etc.). You will probably have to fully qualify the method name (example: **somf_TDequeliterator_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_TSequenceliterator**.

ev

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

Return Value

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

Example

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

Original Class

somf_TIterator Class (overridden here)

Related Information

somfNext Method

somfLast Method

Gets the last object in an ordered collection.

IDL Syntax

```
somf_MCollectible somfLast ( );
```

Description

The **somfLast** method determines the last object in the **somf_TSequence Class** collection that corresponds to the **somf_TSequenceliterator** iterator used as the receiving object, and returns a pointer to it. Every class that inherits from the **somf_TSequenceliterator** class must override this method for that class to work correctly.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. **somfLast** is a method name declared in multiple parents (for example: **somf_TSequenceliterator**, **somf_TSequence**). You will probably have to fully qualify the name of the method (for example:

somf_TSortedSequenceliterator_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_TSequenceliterator**.

ev

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

Return Value

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

Example

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

Original Class

somf_TSequenceliterator

Related Information

somfPrevious Method

somfNext Method

Gets the next object in an ordered collection.

IDL Syntax

```
somf_MCollectible somfNext ( );
```

Description

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

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

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Thus, if any child of **somf_Titerator** is used with a child of **somf_TPrimitiveLinkedListiterator**, then the name of the method will have to be fully qualified (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_TSequenceliterator**.

ev

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

Return Value

- **somf_MCollectible**, a pointer to the next **somf_MCollectible** object in the collection.
- **SOMF_NIL**, the end of the collection has been reached.

Example

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

Original Class

somf_Titerator Class (overridden here)

Related Information

somfFirst Method

sopfPrevious Method

Gets the previous object in an ordered collection.

IDL Syntax

```
sopf_MCollectible sopfPrevious ( );
```

Description

The **sopfPrevious** method determines the previous object in the **sopf_TSequence Class** collection that corresponds to the **sopf_TSequenceliterator** iterator used as the receiving object, and returns a pointer to the previous object (if found). Every class that inherits from the **sopf_TSequenceliterator** class must override this method for that class to work correctly.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Thus, if any child of **sopf_TSequenceliterator** is used with **sopf_TPrimitiveLinkedListliterator**, then the name of the method will have to be fully qualified (example: **sopf_TSortedSequenceliterator_sopfPrevious**). 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->sopfPrevious(ev);
```

Parameters

receiver

A pointer to an object of class **sopf_TSequenceliterator**.

ev

A pointer to the Environment structure for the calling method.

Return Value

- **sopf_MCollectible**, a pointer to the previous **sopf_MCollectible** object in the collection.
- **SOPF_NIL**, the beginning of the collection has been reached.

Example

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

Original Class

sopf_TSequenceliterator

Related Information

sopfLast Method

somfRemove Method

Removes the current object from an ordered collection.

IDL Syntax

```
void somfRemove ( );
```

Description

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

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

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

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

C cannot handle methods from different classes having the same name when they inherit the name from different parents. **somfRemove** is a method name declared in multiple parents (example: **somf_TCollection**, **somf_THashTable**, **somf_TIterator**). You will probably have to fully qualify the method name (for example: **somf_TDictionaryIterator_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_TSequenceliterator**.

ev

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

Example

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

Original Class

somf_TIterator Class (overridden here)

somf_TSet Class

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

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

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

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

The **somf_TSet** class uses **somflsEqual** as the default comparison function. If `key1="Bart"` and `key2="Bart"`, then `key1` and `key2` are equal. If you do not want to use this method to equate entries, use an initialization methods to change to **somflsSame**.

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

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

File Stem

tset

Base

somf_TCollection Class

Metaclass

SOMClass

Ancestor Classes

somf_TCollection Class
 somf_MCollectible Class
 SOMObject

New Methods

somfDifferenceS Method
 somfDifferenceSS Method
 somfIntersectionS Method
 somfIntersectionSS Method
 somfUnionS Method
 somfUnionSS Method
 somfXorS Method
 somfXorSS Method
 somfSetHashFunction Method
 somfGetHashFunction Method

somf_TSet Class

somfRehash Method
somfAssign Method
somfTSetInitFL Method
somfTSetInitF Method
somfTSetInitLF Method
somfTSetInitL Method
somfTSetInitS Method

Overriding Methods

somDefaultInit Method
somDestruct Method
somfAdd Method
somfRemove Method
somfRemoveAll Method
somfDeleteAll Method
somfCount Method
somfMember Method
somfCreateIterator Method

somfAdd Method

Adds an object to a given set.

IDL Syntax

```
somf_MCollectible somfAdd (in somf_MCollectible obj);
```

Description

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

Parameters

receiver

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

ev

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

obj

A pointer to the **somf_MCollectible Class** object that will be added to the set.

Return Value

- **somf_MCollectible**; a pointer to the **somf_MCollectible** object that had to be removed in order to add *obj*.
- **SOMF_NIL**; no **somf_MCollectible** object had to be removed in order to add *obj*.

Example

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

ev = somGetGlobalEnvironment();

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

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

_somFree (s);
```

Original Class

somf_TCollection Class (overridden here)

somfAssign Method

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

IDL Syntax

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

Description

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

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

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

Parameters

receiver

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

ev

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

source

A pointer to the set to which the receiving object will be made equal.

Example

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

ev = somGetGlobalEnvironment();

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

/* Add som objects to s1 */

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

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

Original Class

somf_TSet

somfCount Method

Gets the number of objects in a given set.

IDL Syntax

```
long somfCount ( );
```

Description

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

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

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

Parameters

receiver

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

ev

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

Return Value

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

Example

```
somf_TSet s;
Environment *ev;

ev = somGetGlobalEnvironment();

s = somf_TSetNew();

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

_somFree (s);
```

Original Class

somf_TCollection Class (overridden here)

somfCreateIterator Method

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

IDL Syntax

```
somf_TIterator somfCreateIterator ( );
```

Description

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

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

Parameters

receiver

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

ev

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

Return Value

This method returns a pointer to the new iterator.

Example

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

ev = somGetGlobalEnvironment();

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

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

Original Class

somf_TCollection Class (overridden here)

somfDeleteAll Method

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

IDL Syntax

```
void somfDeleteAll ( );
```

Description

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

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. If multiple pointers to *A* exist, or if a single pointer to *A* is in the collection multiple times, the behavior of the code is undefined, because it will try to delete *A* multiple times. If you think there is a chance that an object could appear in the collection more than once, you should consider using **somfRemoveAll Method** 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_TSet_somfDeleteAll**). This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

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

Parameters

receiver

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

ev

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

Example

```
somf_TSet s;
Environment *ev;

ev = somGetGlobalEnvironment();

s = somf_TSetNew();

/* Add some objects to s */

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

_somFree (s);
```

Original Class

somf_TCollection Class (overridden here)

somfDifferenceS Method

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

IDL Syntax

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

Description

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

Parameters

receiver

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

ev

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

set1

A pointer to the set that the receiving object will be compared against.

Example

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

ev = somGetGlobalEnvironment();

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

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

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

Original Class

somf_TSet

Related Information

somfDifferenceSS Method

somfDifferenceSS Method

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

IDL Syntax

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

Description

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

Parameters

receiver

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

ev

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

set1

A pointer to the set that the receiving object set will be compared against.

resultSet

A pointer to the set containing the results of the operation.

Example

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

ev = somGetGlobalEnvironment();

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

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

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

Original Class

somf_TSet

Related Information

somfDifferenceS Method

somfGetHashFunction Method

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

IDL Syntax

```
somf_MCollectibleHashFn somfGetHashFunction ( );
```

Description

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

C cannot handle methods from different classes having the same name when they inherit the name from different parents. If **somf_TSet** is used with a child of **somf_THashTable** or **somf_TDictionary**, then the name of the method will have to be fully qualified (example: **somf_TSet_somfGetHashFunction**). This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

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

Parameters

receiver

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

ev

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

Return Value

This method returns a pointer to the hash function.

Example

```
somf_TSet s;
Environment *ev;

ev = somGetGlobalEnvironment();

s = somf_TSetNew();

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

_somFree (s);
```

Original Class

somf_TSet

Related Information

somfSetHashFunction Method

somfIntersectionS Method

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

IDL Syntax

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

Description

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

Parameters

receiver

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

ev

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

set1

A pointer to the set that the receiving object will be compared against.

Example

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

ev = somGetGlobalEnvironment();

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

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

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

Original Class

somf_TSet

Related Information

somfIntersectionSS Method

somfIntersectionSS Method

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

IDL Syntax

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

Description

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

Parameters

receiver

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

ev

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

set1

A pointer to the set this instance will be compared against.

resultSet

A pointer to the set containing the results of the operation.

Example

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

ev = somGetGlobalEnvironment();

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

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

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

Original Class

somf_TSet

Related Information

somfIntersectionS Method

somfMember Method

Gets an object from a set.

IDL Syntax

```
somf_MCollectible somfMember (in somf_MCollectible obj);
```

Description

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

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

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

Parameters

receiver

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

ev

A pointer to the Environment structure for the calling method.

obj

A pointer to the **somf_MCollectible** that may or may not be a member of the Collection.

Return Value

- **somf_MCollectible**; a pointer to the object the method determined as the member.
- **SOMF_NIL**; the object was not found.

Example

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

ev = somGetGlobalEnvironment();

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

_somfAdd(s, ev, obj);

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

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

Original Class

somf_TCollection Class (overridden here)

somfRehash Method

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

IDL Syntax

```
void somfRehash ( );
```

Description

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

Note: You cannot override this method.

Parameters

receiver

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

ev

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

Example

```
somf_TSet s;  
Environment *ev;  
  
ev = somGetGlobalEnvironment();  
  
s = somf_TSetNew();  
  
_somfRehash(s, ev);  
  
_somFree (s);
```

Original Class

somf_TSet

somfRemove Method

Removes an object from a given set.

IDL Syntax

```
somf_MCollectible somfRemove (in somf_MCollectible obj);
```

Description

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

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

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

Parameters

receiver

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

ev

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

obj

A pointer to the **somf_MCollectible** object to be removed from the set.

Return Value

- **somf_MCollectible**; a pointer to the object that was removed.
- **SOMF_NIL**; the object was not found.

Example

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

ev = somGetGlobalEnvironment();

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

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

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

Original Class

somf_TCollection Class (overridden here)

Related Information

somfRemoveAll Method

somfRemoveAll Method

Removes all of the objects from a given set.

IDL Syntax

```
void somfRemoveAll ( );
```

Description

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

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

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

Parameters

receiver

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

ev

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

Example

```
somf_TSet s;
Environment *ev;

ev = somGetGlobalEnvironment();

s = somf_TSetNew();

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

_somFree (s);
```

Original Class

somf_TCollection Class (overridden here)

Related Information

somfRemove Method

somfSetHashFunction Method

Sets the hash function of a set.

IDL Syntax

```
void somfSetHashFunction (in somf_MCollectibleHashFn fn);
```

Description

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

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

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

Parameters

receiver

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

ev

A pointer to the Environment structure for the calling method.

fn

A function pointer specifying a **somfHash Method** type function.

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 Class**. The **somf_TSet** object will use this pointer to access the **somfHash** method that was declared and defined in the object being inserted into, or removed from, the **somf_TSet** object.

Example

```
somf_TSet s;
Environment *ev;

ev = somGetGlobalEnvironment();

s = somf_TSetNew();

somf_TSet_somfSetHashFunction(s, ev,
                              somf_MCollectibleClassData.somfHash);

_somFree (s);
```

Original Class

somf_TSet

Related Information

somfGetHashFunction Method

somfTSetInitF Method

Initializes a new set, given its comparison test method.

IDL Syntax

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

Description

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

Note: You cannot override this method.

Parameters

receiver

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

ev

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

testfn

A method pointer specifying either a **somflsEqual** or a **somflsSame** 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 Class**. The **somf_TSet** object will use this pointer to access the **somflsSame Method** or **somflsEqual Method** that was declared and defined in the object being inserted into, or removed from, the **somf_TSet** object.

Return Value

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

Example

```
somf_TSet s2;
Environment *ev;

ev = somGetGlobalEnvironment();

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

_somFree (s2);
```

Original Class

somf_TSet

Related Information

somfTSetInitFL Method
somfTSetInitLF Method
somfTSetInitL Method
somfTSetInitS Method

somfTSetInitFL Method

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

IDL Syntax

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

Description

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

Note: You cannot override this method.

Parameters

receiver

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

ev

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

testfn

A method pointer specifying either a **somflsEqual** or a **somflsSame** method.

This argument should always be set to either

```
somf_MCollectibleClassData.somflsSame or
somf_MCollectibleClassData.somflsEqual.
```

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

setSizeHint

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

Return Value

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

Example

```
somf_TSet s1;
Environment *ev;
ev = somGetGlobalEnvironment();
s1 = somf_TSetNew();
_somfTSetInitFL(s1, ev, somf_MCollectibleClassData.somflsEqual, 8);
_somFree (s1);
```

Original Class

somf_TSet

Related Information

somfTSetInitLF Method
somfTSetInitF Method
somfTSetInitL Method
somfTSetInitS Method

somfTSetInitL Method

Initializes a new set, given the initial set size.

IDL Syntax

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

Description

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

Note: You cannot override this method.

Parameters

receiver

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

ev

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

setSizeHint

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

Return Value

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

Example

```
somf_TSet s4;
Environment *ev;

ev = somGetGlobalEnvironment();

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

_somFree (s4);
```

Original Class

somf_TSet

Related Information

somfTSetInitFL Method
somfTSetInitLF Method
somfTSetInitF Method
somfTSetInitS Method

somfTSetInitLF Method

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

IDL Syntax

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

Description

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

Note: You cannot override this method.

Parameters

receiver

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

ev

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

testfn

A method pointer specifying either a **somflsEqual** or a **somflsSame** 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 Class**. The **somf_TSet** object will use this pointer to access the **somflsSame Method** or **somflsEqual Method** that was declared and defined in the object being inserted into, or removed from, the **somf_TSet** object.

Return Value

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

Example

```
somf_TSet s3;
Environment *ev;

ev = somGetGlobalEnvironment();

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

_somFree (s3);
```

Original Class

somf_TSet

Related Information

somfTSetInitFL Method
somfTSetInitF Method
somfTSetInitL Method
somfTSetInitS Method

somfTSetInitS Method

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

IDL Syntax

```
somf_TSet somfTSetInitS (in somf_TSet s);
```

Description

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

Note: You cannot override this method.

Parameters

receiver

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

ev

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

s

A pointer to the set to which the receiving object will be equal.

Return Value

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

Example

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

ev = somGetGlobalEnvironment();

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

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

Original Class

somf_TSet

Related Information

somfTSetInitFL Method
somfTSetInitLF Method
somfTSetInitF Method
somfTSetInitL Method

somfUnionS Method

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

IDL Syntax

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

Description

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

Parameters

receiver

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

ev

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

set1

A pointer to the set that the receiving object will be compared against.

Example

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

ev = somGetGlobalEnvironment();

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

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

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

Original Class

somf_TSet

Related Information

somfUnionSS Method

somfUnionSS Method

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

IDL Syntax

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

Description

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

Parameters

receiver

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

ev

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

set1

A pointer to the set that the receiving object will be compared against.

resultSet

A pointer to the set containing the results of the operation.

Example

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

ev = somGetGlobalEnvironment();

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

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

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

Original Class

somf_TSet

Related Information

somfUnionS Method

somfXorS Method

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

IDL Syntax

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

Description

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

Parameters

receiver

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

ev

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

set1

A pointer to the set that the receiving object will be compared against.

Example

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

ev = somGetGlobalEnvironment();

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

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

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

Original Class

somf_TSet

Related Information

somfXorSS Method

somfXorSS Method

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

IDL Syntax

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

Description

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

Parameters

receiver

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

ev

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

set1

A pointer to the set that the receiving object will be compared against.

resultSet

A pointer to the set containing the results of the operation.

Example

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

ev = somGetGlobalEnvironment();

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

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

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

Original Class

somf_TSet

Related Information

somfXorS Method

somf_TSetIterator Class

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

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

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

File Stem

tsetitr

Base

somf_TIterator Class

Metaclass

SOMClass

Ancestor Classes

somf_TIterator Class
SOMObject

New Methods

somfTSetIteratorInit Method

Overriding Methods

somDestruct Method
somfNext Method
somfFirst Method
somfRemove Method

somfFirst Method

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

IDL Syntax

```
somf_MCollectible somfFirst ();
```

Description

The **somfFirst** method resets the iterator and returns the first element of the set that corresponds to the set iterator represented by the receiving object. **somfFirst** resets the iterator to the beginning of the set even if other operations on the collection cause the iterator to be invalidated. In the second case, the method 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. You will probably have to fully qualify the method name. This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you can referenced this method as:

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

Parameters

receiver

A pointer to an object of class **somf_TSetIterator**.

ev

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

Return Value

- a pointer to the first **somf_MCollectible** object in the set.
- SOMF_NIL is returned if the collection is empty.

Example

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

ev = somGetGlobalEnvironment();

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

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

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

Original Class

somf_TIterator Class

Related Information

somfNext Method

somfNext Method

Gets the next object in a set.

IDL Syntax

```
somf_MCollectible somfNext ();
```

Description

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

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

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

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

Parameters

receiver

A pointer to an object of class **somf_TSetIterator**.

ev

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

Return Value

- **somf_MCollectible**; a pointer to the next **somf_MCollectible** object in the set.
- **SOMF_NIL**; the end of the set has been reached.

Example

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

ev = somGetGlobalEnvironment();

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

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

somfNext Method

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

Original Class

somf_Tliterator Class

Related Information

somfFirst Method

somfRemove Method

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

IDL Syntax

```
void somfRemove ();
```

Description

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

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

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

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

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

Parameters

receiver

A pointer to an object of class **somf_TSetIterator**.

ev

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

Example

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

ev = somGetGlobalEnvironment();

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

/* Remove the first object in s */

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

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

Original Class

somf_TIterator Class

somfTSetIteratorInit Method

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

IDL Syntax

```
somf_TSetIterator somfTSetIteratorInit (in somf_TSet h);
```

Description

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

This is one of two ways to initialize a **somf_TSetIterator** to point to an instance of a **somf_TSet** set. The other way is to use the **somf_TSet** class's **somfCreateIterator Method**.

Note: You cannot override this method.

Parameters

receiver

A pointer to an object of class **somf_TSetIterator**.

ev

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

h

A pointer to the set that the receiving object will iterate over.

Return Value

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

Example

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

ev = somGetGlobalEnvironment();

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

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

Original Class

somf_TSetIterator

somf_TSortedSequence Class

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

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

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

All **somf_MOrderableCollectible** objects inserted into a **somf_TSortedSequence** collection should override **somflsEqual**, **somflsLessThan**, **somflsGreaterThan** and **somfHash**.

The **somf_TSortedSequence** class uses **somflsEqual** to compare objects in the collection. You cannot override or change this to the **somflsSame**.

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

File Stem

tss

Base

somf_TSequence Class

Metaclass

SOMClass Class

Ancestor Classes

somf_TSequence Class
 somf_TCollection Class
 somf_MCollectible Class
 SOMObject Class

New Methods

somfAssign Method
 somfCreateSequenceIterator Method
 somfCreateSortedSequenceNode Method
 somfGetSequencingFunction Method
 somfSetSequencingFunction Method
 somfTSortedSequenceInitF Method
 somfTSortedSequenceInitS Method

Overriding Methods

somDefaultInit Method
 somDestruct Method
 somfAdd Method
 somfAfter Method
 somfBefore Method

somf_TSortedSequence Class

somfCount Method
somfCreateIterator Method
somfDeleteAll Method
somfFirst Method
somfLast Method
somfMember Method
somfOccurrencesOf Method
somfRemove Method
somfRemoveAll Method

somfAdd Method

Adds an *obj* to a sorted sequence.

IDL Syntax

```
somf_MCollectible somfAdd (in somf_MCollectible obj);
```

Description

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

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

Parameters

receiver

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

ev

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

obj

A pointer to an **somf_MCollectible** that will be added to this instance.

Return Value

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

Example

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

ev = somGetGlobalEnvironment();

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

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

_somFree (ss);
```

Original Class

somf_TCollection Class (overridden here)

somfAfter Method

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

IDL Syntax

```
somf_MCollectible somfAfter (in somf_MCollectible obj);
```

Description

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

Parameters

receiver

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

ev

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

obj

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

Return Value

- **somf_MCollectible**; A pointer to the **somf_MCollectible** object after *obj*.
- **SOMF_NIL**; The designated *obj* is the last object in this collection, or not found.

Example

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

ev = somGetGlobalEnvironment();

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

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

_somFree (ss);
```

Original Class

somf_TSortedSequence Class (overridden here)

Related Information

somfBefore Method
somfFirst Method
somfLast Method

somfAssign Method

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

IDL Syntax

```
void somfAssign (in somf_TSortedSequence s);
```

Description

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

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

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

Parameters

receiver

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

ev

A pointer to the Environment structure for the calling method.

s

A pointer to the sorted sequence to which the receiver will be equal.

Example

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

ev = somGetGlobalEnvironment();

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

/* Add som objects to s1 */

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

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

Original Class

somf_TSortedSequence Class

somfBefore Method

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

IDL Syntax

```
somf_MCollectible somfBefore (in somf_MCollectible obj);
```

Description

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

Parameters

receiver

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

ev

A pointer to the Environment structure for the calling method.

obj

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

Return Value

- **somf_MCollectible**; A pointer to the **somf_MCollectible** object that precedes *obj*.
- SOMF_NIL; the designated *obj* is the first object in the sequence, or not found.

Example

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

ev = somGetGlobalEnvironment();
ss = somf_TSortedSequenceNew();
obj =
    <Your class inheriting from somf_MOrderableCollectible>New();

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

_somFree (ss);
```

Original Class

somf_TSequence Class (overridden here)

Related Information

somfAfter Method
somfFirst Method
somfLast Method

somfCount Method

Gets the number of objects in a given sorted sequence.

IDL Syntax

```
long somfCount ( );
```

Description

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

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

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

Parameters

receiver

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

ev

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

Return Value

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

Example

```
somf_TSortedSequence ss;
Environment *ev;

ev = somGetGlobalEnvironment();
ss = somf_TSortedSequenceNew();

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

_somFree (ss);
```

Original Class

somf_TCollection Class (overridden here)

somfCreateIterator Method

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

IDL Syntax

```
somf_TIterator somfCreateIterator ( );
```

Description

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

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

Parameters

receiver

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

ev

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

Return Value

This method returns a pointer to the new iterator.

Example

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

ev = somGetGlobalEnvironment();

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

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

Original Class

somf_TCollection Class (overridden here)

Related Information

somfCreateSequenceIterator Method

somfCreateSequenceliterator Method

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

IDL Syntax

```
somf_TSequenceliterator somfCreateSequenceliterator ( );
```

Description

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

Note: This is one of three ways to initialize a **somf_TSortedSequenceliterator Class** to point to an instance of a **somf_TSortedSequence**. One other way is to use the initializer method for the **somf_TSortedSequenceliterator** class. The final way is to use the **somf_TSortedSequence** class's **somfCreateliterator**.

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

Parameters

receiver

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

ev

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

Return Value

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

Example

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

ev = somGetGlobalEnvironment();

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

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

Original Class

somf_TSortedSequence

Related Information

somfCreateliterator Method

somfCreateSortedSequenceNode Method

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

IDL Syntax

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

Description

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

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

Parameters

receiver

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

ev

A pointer to the Environment structure for the calling method.

n1

A pointer to the left child of the new **somf_TSortedSequenceNode** object.

ob

A pointer to the key of the new **somf_TSortedSequenceNode** object.

n2

A pointer to the right child of the new **somf_TSortedSequenceNode** object.

Return Value

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

Original Class

somf_TSortedSequence Class

somfDeleteAll Method

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

IDL Syntax

```
void somfDeleteAll ( );
```

Description

The **somfDeleteAll** method removes all of the objects from the sorted sequence represented by the receiving object. The method deallocates the storage that these objects might have owned. 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 Method** 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_TSortedSequence**.

ev

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

Example

```
somf_TSortedSequence ss;
Environment *ev;
ev = somGetGlobalEnvironment();
ss = somf_TSortedSequenceNew();
/* Remove all the objects from ss AND DELETE THEM */
somf_TSortedSequence_somfDeleteAll(ss, ev);
_somFree (ss);
```

Original Class

somf_TCollection Class (overridden here)

somfFirst Method

Gets the first object in a sorted sequence.

IDL Syntax

```
somf_MCollectible somfFirst ( );
```

Description

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

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

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

Parameters

receiver

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

ev

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

Return Value

- **somf_MCollectible**; a pointer to the first **somf_MCollectible** object in the sorted sequence.
- **SOMF_NIL**; nothing is in the collection.

Example

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

ev = somGetGlobalEnvironment();

ss = somf_TSortedSequenceNew();

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

_somFree (ss);
```

Original Class

somf_TSequence Class (overridden here)

Related Information

somfAfter Method
somfBefore Method
somfLast Method

somfGetSequencingFunction Method

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

IDL Syntax

```
somf_MBetterOrderableCompareFn somfGetSequencingFunction ( );
```

Description

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

Parameters

receiver

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

ev

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

Return Value

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

Example

```
somf_TSortedSequence ss;
Environment *ev;
ev = somGetGlobalEnvironment();
ss = somf_TSortedSequenceNew();
if (_somfGetSequencingFunction(ss, ev) !=
    somf_MOrderableCollectibleClassData.somfCompare)
{
    somPrintf("\n What Compare Function are we using?\n");
}
_somFree (ss);
```

Original Class

somf_TSortedSequence

Related Information

somfSetSequencingFunction Method

somfLast Method

Gets the last object in a sorted sequence.

IDL Syntax

```
somf_MCollectible somfLast ( );
```

Description

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

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

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

Parameters

receiver

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

ev

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

Return Value

- **somf_MCollectible**; a pointer to the last **somf_MCollectible** object in the collection.
- **SOMF_NIL**; Nothing is in the collection.

Example

```
somf_TSortedSequence ss;
Environment *ev;
somf_MOrderableCollectible obj;
ev = somGetGlobalEnvironment();
ss = somf_TSortedSequenceNew();
/* Determine the last object in ss */
obj = somf_TSortedSequence_somfLast(ss, ev);

_somFree (ss);
```

Original Class

somf_TSequence Class (overridden here)

Related Information

somfAfter Method
somfBefore Method
somfFirst Method

somfMember Method

Gets an object in a sorted sequence.

IDL Syntax

```
somf_MCollectible somfMember (in somf_MCollectible obj);
```

Description

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

C cannot handle methods from different classes having the same name when they inherit the name from different parents. If any child of **somf_TCollection** is used with **somf_THashTable**, then the name of the method will have to be fully qualified, so the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

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

Parameters

receiver

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

ev

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

obj

A pointer to the **somf_MCollectible** object that may be a member.

Return Value

- **somf_MCollectible**; a pointer to the object the method determined as a member.
- **SOMF_NIL**; the object was not found.

Example

```
somf_TSortedSequence ss;
<your Class inheriting from somf_MOrderableCollectible> obj;
Environment *ev;
ev = somGetGlobalEnvironment();
ss = somf_TSortedSequenceNew();
obj =
  <your Class inheriting from somf_MOrderableCollectible>New();
_somfAdd(ss, ev, obj);
if (_somfMember(ss, ev, obj) != SOMF_NIL)
  somPrintf("\n obj is a Member\n");
else
  somPrintf("\n ERROR: obj should be a Member\n");
_somFree (ss);
_somFree (obj);
```

Original Class

somf_TCollection Class (overridden here)

Related Information

somfOccurrencesOf Method

somfOccurrencesOf Method

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

IDL Syntax

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

Description

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

Parameters

receiver

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

ev

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

obj

A pointer to the **somf_MCollectible** to look for in the collection.

Return Value

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

Example

```
somf_TSortedSequence ss;
<your Class which inherits from somf_MOrderableCollectible> obj;
Environment *ev;
ev = somGetGlobalEnvironment();
ss = somf_TSortedSequenceNew();
obj = <your Class which inherits from
        somf_MOrderableCollectible>New();
somPrintf("\n There are %d OccurrencesOf obj\n",
        _somfOccurrencesOf(ss, ev, obj));
_somFree (ss);
_somFree (obj);
```

Original Class

somf_TSequence Class (overridden here)

Related Information

somfMember Method

somfRemove Method

Removes an object from a sorted sequence.

IDL Syntax

```
somf_MCollectible somfRemove (in somf_MCollectible obj);
```

Description

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

C cannot handle methods from different classes having the same name when they inherit the name from different parents. **somfRemove** is a method name declared in multiple parents. You will probably have to fully qualify the method name. This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

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

Parameters

receiver

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

ev

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

obj

A pointer to the **somf_MCollectible** object to be removed from the collection.

Return Value

There are two possible valid return values for this method:

somf_MCollectible

A pointer to the object that was actually removed.

SOMF_NIL The specified object was not found.

Example

```
somf_TSortedSequence ss;
<your Class which inherits from somf_MOrderableCollectible> obj;
Environment *ev;
ev = somGetGlobalEnvironment();
ss = somf_TSortedSequenceNew();
obj = <your Class which inherits from
      somf_MOrderableCollectible>New();
/* Remove obj from ss */
somf_TSortedSequence_somfRemove(ss, ev, obj);
_somFree (ss);
_somFree (obj);
```

Original Class

somf_TCollection Class (overridden here)

Related Information

somfRemoveAll Method

somfRemoveAll Method

Removes all of the objects from a sorted sequence.

IDL Syntax

```
void somfRemoveAll ( );
```

Description

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

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Thus, if any child of **somf_TCollection** is used with **somf_THashTable**, then the name of the method will have to be fully qualified. This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

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

Parameters

receiver

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

ev

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

Example

```
somf_TSortedSequence ss;
Environment *ev;
ev = somGetGlobalEnvironment();
ss = somf_TSortedSequenceNew();
/* Remove all the objects from ss */
somf_TSortedSequence_somfRemoveAll(ss, ev);
_somFree (ss);
```

Original Class

somf_TCollection Class (overridden here)

Related Information

somfRemove Method

somfSetSequencingFunction Method

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

IDL Syntax

```
void somfSetSequencingFunction (in somf_MBetterOrderableCompareFn fn);
```

Description

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

Parameters

receiver

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

ev

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

fn

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

This should always be set to:

```
somf_MOrderableCollectibleClassData.somfCompare.
```

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

Example

```
somf_TSortedSequence ss;
Environment *ev;
ev = somGetGlobalEnvironment();
ss = somf_TSortedSequenceNew();
_somfSetSequencingFunction(ss, ev,
    somf_MOrderableCollectibleClassData.somfCompare);
_somFree (ss);
```

Original Class

somf_TSortedSequence Class

Related Information

somfGetSequencingFunction Method

somfTSortedSequenceInitF Method

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

IDL Syntax

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

Description

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

Note: You cannot override this method.

Parameters

receiver

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

ev

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

testfn

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

This should always be set to:

```
somf_MOrderableCollectibleClassData.somfCompare.
```

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

Return Value

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

Example

```
somf_TSortedSequence s1;
Environment *ev;
ev = somGetGlobalEnvironment();
s1 = somf_TSortedSequenceNew();
_somfTSortedSequenceInitF(s1, ev,
    somf_MOrderableCollectibleClassData.somfCompare);
_somFree (s1);
```

Original Class

somf_TSortedSequence Class

Related Information

somfTSortedSequenceInitS Method

somfTSortedSequenceInitS Method

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

IDL Syntax

```
somf_TSortedSequence somfTSortedSequenceInitS (
    in somf_TSortedSequence s);
```

Description

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

Note: You cannot override this method.

Parameters

receiver

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

ev

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

s

A pointer to the **somf_TSortedSequence** object to which the new sorted sequence will be equal.

Return Value

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

Example

```
somf_TSortedSequence s1;
somf_TSortedSequence s2;
Environment *ev;
ev = somGetGlobalEnvironment();
s1 = somf_TSortedSequenceNew();
s2 = somf_TSortedSequenceNew();
_somfTSortedSequenceInitS(s2, ev, s1);
_somFree (s1);
_somFree (s2);
```

Original Class

somf_TSortedSequence Class

Related Information

somfTSortedSequenceInitF Method

somf_TSortedSequenceliterator Class

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

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

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

File Stem

tssitr

Base

somf_TSequenceliterator Class

Metaclass

SOMClass Class

Ancestor Classes

somf_TSequenceliterator Class

somf_Titerator Class

SOMObject Class

New Methods

somfStartHere Method

somfTSortedSequenceliteratorInit Method

Overriding Methods

somfFirst Method

somfLast Method

somfNext Method

somfPrevious Method

somfRemove Method

somfFirst Method

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

IDL Syntax

```
somf_MCollectible somfFirst ( );
```

Description

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

This method resets the iterator to the beginning of the sorted sequence even when other operations on the collection cause the iterator to be invalidated. In the second case, this method 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. You will probably have to fully qualify the name of the method. This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

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

Parameters

receiver

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

ev

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

Return Value

- a pointer to the first **somf_MCollectible** object in the sorted sequence
- SOMF_NIL is returned if the collection is empty.

Example

```
somf_TSortedSequence ss;
Environment *ev;
somf_TSortedSequenceIterator itr;
<Your class inheriting from somf_MOrderableCollectible> itrobj;
ev = somGetGlobalEnvironment();
ss = somf_TSortedSequenceNew();
itr = somf_TSortedSequenceIteratorNew();
_somfTSortedSequenceIteratorInit(itr, ev, ss);
/* Iterate through the TSortedSequence */
itrobj =
    (<Your class that inherits from somf_MOrderableCollectible>*)
    somf_TSortedSequenceIterator_somfFirst(itr, ev);
while (itrobj != SOMF_NIL)
/* Do something with itrobj */
{itrobj =
    (<Your class inheriting from somf_MOrderableCollectible>*)
    _somfNext(itr, ev);}
_somFree (ss);
_somFree (itr);
```

Original Class

somf_TIterator Class (overridden here)

somfFirst Method

Related Information

somfNext Method

somfStartHere Method

somfLast Method

Gets the last object in a sorted sequence.

IDL Syntax

```
somf_MCollectible somfLast ( );
```

Description

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

C cannot handle methods from different classes having the same name when they inherit the name from different parents. **somfLast** is a method name declared in multiple parents (for example: **somf_TSequenceIterator**, **somf_TSequence**). You will probably have to fully qualify the name of the method. This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

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

Parameters

receiver

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

ev

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

Return Value

- a pointer to the last **somf_MCollectible** object in the sorted sequence.
- SOMF_NIL is returned if the collection is empty.

Example

```
somf_TSortedSequence ss;
Environment *ev;
somf_TSortedSequenceIterator itr;
<Your class inheriting from somf_MOrderableCollectible> itrobj;
ev = somGetGlobalEnvironment();
ss = somf_TSortedSequenceNew();
itr = somf_TSortedSequenceIteratorNew();
_somfTSortedSequenceIteratorInit(itr, ev, ss);
/* Go to the last object in ss */
itrobj =
    (<Your class which inherits from somf_MOrderableCollectible>*)
    somf_TSortedSequenceIterator_somfLast(itr, ev);
_somFree (ss);
_somFree (itr);
```

Original Class

somf_TSequenceIterator (overridden here)

Related Information

somfPrevious Method

somfNext Method

Gets the next object in a sorted sequence.

IDL Syntax

```
somf_MCollectible somfNext ();
```

Description

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

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

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Thus, if any child of **somf_TIterator** is used with a child of **somf_TPrimitiveLinkedListIterator**, then the name of the method will have to be fully qualified. This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

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

Parameters

receiver

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

ev

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

Return Value

- **somf_MCollectible**; a pointer to the next **somf_MCollectible** object in the sorted sequence.
- **SOMF_NIL**; the end of the collection has been reached.

Example

```
somf_TSortedSequence ss;
Environment *ev;
somf_TSortedSequenceIterator itr;
<Your class inheriting from somf_MOrderableCollectible> itrobj;
ev = somGetGlobalEnvironment();
ss = somf_TSortedSequenceNew();
itr = somf_TSortedSequenceIteratorNew();
_somfTSortedSequenceIteratorInit(itr, ev, ss);
/* Iterate through the TSortedSequence */
itrobj =
    (<Your class inheriting from somf_MOrderableCollectible>*)
    somf_TSortedSequenceIterator_somfFirst(itr, ev);
while (itrobj != SOMF_NIL)
{
    /* Do something with itrobj */
}
```

```
        itrobj =  
        (<Your class inheriting from somf_MOrderableCollectible>*)  
            _somfNext(itr, ev);  
    }  
    _somFree (ss);  
    _somFree (itr);
```

Original Class

somf_TIterator Class (overridden here)

Related Information

somfFirst Method

somfStartHere Method

somfPrevious Method

Gets the previous object in a sorted sequence.

IDL Syntax

```
somf_MCollectible somfPrevious ( );
```

Description

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

If the **somf_TSortedSequence** collection has changed since **somfLast** was called, the iterator becomes invalid and will fail if asked to find the previous object. If the collection's **somfAdd** method were called after starting to iterate through the collection, the iterator then would not allow iteration to continue. The iterator must be reset. The easiest way is to call the iterator's **somfLast** method and restart.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. If any child of **somf_TSequenceIterator** is used with **somf_TPrimitiveLinkedListIterator**, the name of the method will have to be fully qualified, so the linker can tell them apart. This is not a problem in C++. In C++ you can referenced this method as:

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

Parameters

receiver

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

ev

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

Return Value

- **somf_MCollectible**; a pointer to the previous object in the sorted sequence collection.
- **SOMF_NIL**; the beginning of the collection has been reached.

Example

```
somf_TSortedSequence ss;
Environment *ev;
somf_TSortedSequenceIterator itr;
<Your class inheriting from somf_MOrderableCollectible> itrobj;
ev = somGetGlobalEnvironment();
ss = somf_TSortedSequenceNew();
itr = somf_TSortedSequenceIteratorNew();
_somfTSortedSequenceIteratorInit(itr, ev, ss);
/* Go to the next to the last object in ss */
somf_TSortedSequenceIterator_somfLast(itr, ev);
itrobj =
    (<Your class which inherits from somf_MOrderableCollectible>*)
        _somfPrevious(itr, ev);
_somFree (ss);
_somFree (itr);
```

Original Class

somf_TSortedSequenceIterator Class (overridden here)

Related Information

somfLast Method

somfRemove Method

Removes the current object just returned by a previous method from a sorted sequence.

IDL Syntax

```
void somfRemove ();
```

Description

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

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

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

C cannot handle methods from different classes having the same name when they inherit the name from different parents. **somfRemove** is a method name declared in multiple parents. You will probably have to fully qualify the method name. This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

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

Parameters

receiver

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

ev

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

Example

```
somf_TSortedSequence ss;
Environment *ev;
somf_TSortedSequenceIterator itr;
<Your class inheriting from somf_MOrderableCollectible> itrobj;
ev = somGetGlobalEnvironment();
ss = somf_TSortedSequenceNew();
itr = somf_TSortedSequenceIteratorNew();
_somfTSortedSequenceIteratorInit(itr, ev, ss);
/* Use the Iterator's Remove to remove the next to last object */
itrobj =
    (<Your class which inherits from somf_MOrderableCollectible>*)
    somf_TSortedSequenceIterator_somfLast(itr, ev);
itrobj =
    (<Your class which inherits from somf_MOrderableCollectible>*)
    _somfPrevious(itr, ev);
somf_TSortedSequenceIterator_somfRemove(itr, ev);
_somFree (ss);
_somFree (itr);
```

Original Class

somf_TIterator Class (overridden here)

somfStartHere Method

Begins Iterating through a **somf_TSortedSequence Class**, starting at a given object.

IDL Syntax

```
somf_MOrderableCollectible somfStartHere (in somf_MOrderableCollectible obj);
```

Description

The **somfStartHere** method begins Iterating through a **somf_TSortedSequence** collection that corresponds to the **somf_TSortedSequenceIterator** iterator used as the receiving object. Iteration begins at the given object.

Parameters

receiver

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

ev

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

obj

A pointer to the **somf_MOrderableCollectible** object where iteration will begin.

Return Value

- a pointer to the **somf_MCollectible** object where iteration started
- SOMF_NIL is returned if the collection is empty.

Example

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

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

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

Original Class

somf_TSortedSequenceIterator Class

Related Information

somfFirst Method
somfNext Method

somfTSortedSequenceIteratorInit Method

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

IDL Syntax

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

Description

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

This is one of three ways to initialize a **somf_TSortedSequenceIterator** to point to an instance of a **somf_TSortedSequence**. The other two ways are:

- to use the **somf_TSortedSequence** class's **somfCreateSequenceIterator** method.
- to use **somf_TSortedSequence**'s **somfCreateIterator Method**.

Note: You cannot override this method.

Parameters

receiver

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

ev

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

h

A pointer to the sorted sequence that the receiving object will iterate over.

Return Value

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

Example

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

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

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

Original Class

somf_TSortedSequenceIterator Class

somf_TSortedSequenceNode Class

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

somf_TSortedSequenceNode class and methods are for creating a new class that:

- needs linkable nodes between objects of the class
- inherits from **somf_TSortedSequence**, and it would be appropriate to override some methods of the **somf_TSortedSequence** class to define additional functionality for those methods.

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

This class is not thread-safe, even with semaphores around the calls, different tasks should not be setting the values of the node. That situation is too prone to having multiple tasks setting conflicting values, leaving the instance in an unacceptable state.

This class is reentrant.

File Stem

tssnode

Base

SOMObject Class

Metaclass

SOMClass Class

Ancestor Classes

SOMObject Class

New Methods

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

Overriding Method

somDefaultInit Method

somfGetKey Method

Gets the key to a node.

IDL Syntax

```
somf_MOrderableCollectible somfGetKey ( );
```

Description

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

Note: You cannot override this method.

Parameters

receiver

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

ev

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

Return Value

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

Original Class

somf_TSortedSequenceNode

Related Information

somfSetKey Method

somfGetLeftChild Method

Gets the left child of a node.

IDL Syntax

```
somf_TSortedSequenceNode somfGetLeftChild ( );
```

Description

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

Note: You cannot override this method.

Parameters

receiver

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

ev

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

Return Value

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

Original Class

somf_TSortedSequenceNode

Related Information

somfSetLeftChild Method

somfGetParent Method

Gets the parent of a node.

IDL Syntax

```
somf_TSortedSequenceNode somfGetParent ( );
```

Description

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

Note: You cannot override this method.

Parameters

receiver

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

ev

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

Return Value

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

Original Class

somf_TSortedSequenceNode

Related Information

somfSetParent Method

somfGetRed Method

Determines whether a node is red or black.

IDL Syntax

```
boolean somfGetRed ( );
```

Description

The **somfGetRed** method determines whether the node represented by the receiving object is red or black.

Note: For a discussion of Red-Black Trees, you can refer to *Algorithms in C++* by Robert Sedgwick (Addison-Wesley Publishing Company, 1992).

Note: You cannot override this method.

Parameters

receiver

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

ev

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

Return Value

- TRUE, the node is red.
- FALSE, the node is black.

Original Class

somf_TSortedSequenceNode

Related Information

somfSetRed Method

somfSetRedOn Method

somfGetRightChild Method

Gets the right child of a node.

IDL Syntax

```
somf_TSortedSequenceNode somfGetRightChild ( );
```

Description

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

Note: You cannot override this method.

Parameters

receiver

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

ev

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

Return Value

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

Original Class

somf_TSortedSequenceNode

Related Information

somfSetRightChild Method

somfSetKey Method

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

IDL Syntax

```
void somfSetKey (in somf_MOrderableCollectible k);
```

Description

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

Note: You cannot override this method.

Parameters

receiver

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

ev

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

k

A pointer to the **somf_MOrderableCollectible** key.

Original Class

somf_TSortedSequenceNode

Related Information

somfGetKey Method

somfSetLeftChild Method

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

IDL Syntax

```
void somfSetLeftChild (in somf_TSortedSequenceNode n);
```

Description

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

Note: You cannot override this method.

Parameters

receiver

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

ev

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

n

A pointer to the left child of the node.

Original Class

somf_TSortedSequenceNode

Related Information

somfGetLeftChild Method

somfSetParent Method

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

IDL Syntax

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

Description

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

Note: You cannot override this method.

Parameters

receiver

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

ev

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

n

A pointer to the parent node of the receiving-object node.

Original Class

somf_TSortedSequenceNode

Related Information

somfGetParent Method

somfSetRed Method

Sets a node to red or black.

IDL Syntax

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

Description

The **somfSetRed** method sets the node represented by the receiving object to either red or black, as determined by the boolean argument.

Note: For a discussion of Red-Black Trees, you can refer to *Algorithms in C++* by Robert Sedgwick (Addison-Wesley Publishing Company, 1992).

Note: You cannot override this method.

Parameters

receiver

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

ev

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

on

One of these two choices:

- TRUE, the node is red.
- FALSE, the node is black.

Original Class

somf_TSortedSequenceNode

Related Information

somfSetRedOn Method

somfGetRed Method

somfSetRedOn Method

Sets a node to red.

IDL Syntax

```
void somfSetRedOn ( );
```

Description

The **somfSetRedOn** method sets the node represented by the receiving object to red, unconditionally.

Note: For a discussion of Red-Black Trees, you can refer to *Algorithms in C++* by Robert Sedgwick (Addison-Wesley Publishing Company, 1992).

Note: You cannot override this method.

Parameters

receiver

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

ev

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

Original Class

somf_TSortedSequenceNode

Related Information

somfSetRed Method

somfGetRed Method

somfSetRightChild Method

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

IDL Syntax

```
void somfSetRightChild (in somf_TSortedSequenceNode n);
```

Description

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

Note: You cannot override this method.

Parameters

receiver

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

ev

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

n

A pointer to the right child of the node.

Original Class

somf_TSortedSequenceNode

Related Information

somfGetRightChild Method

somfTSortedSequenceNodeInitT Method

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

IDL Syntax

```
somf_TSortedSequenceNode somfTSortedSequenceNodeInitT (
    in somf_TSortedSequenceNode n1);
```

Description

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

Note: You cannot override this method.

Parameters

receiver

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

ev

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

n1

A pointer to the left child of the new **somf_TSortedSequenceNode** object.

Return Value

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

Original Class

somf_TSortedSequenceNode

Related Information

somfTSortedSequenceNodeInitTMT Method

somfTSortedSequenceNodeInitTM Method

somfTSortedSequenceNodeInitTM Method

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

IDL Syntax

```
somf_TSortedSequenceNode somfTSortedSequenceNodeInitTM (  

    in somf_TSortedSequenceNode n1,  

    in somf_MOrderableCollectible obj);
```

Description

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

Note: You cannot override this method.

Parameters

receiver

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

ev

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

n1

A pointer to the left child of the new **somf_TSortedSequenceNode** object.

obj

A pointer to the key of the new **somf_TSortedSequenceNode** object.

Return Value

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

Original Class

somf_TSortedSequenceNode

Related Information

somfTSortedSequenceNodeInitTMT Method

somfTSortedSequenceNodeInitT Method

somfTSortedSequenceNodeInitTMT Method

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

IDL Syntax

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

Description

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

Note: You cannot override this method.

Parameters

receiver

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

ev

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

n1

A pointer to the left child of the new **somf_TSortedSequenceNode** object.

obj

A pointer to the key of the new **somf_TSortedSequenceNode** object.

n2

A pointer to the right child of the new **somf_TSortedSequenceNode** object.

Return Value

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

Original Class

somf_TSortedSequenceNode

Related Information

somfTSortedSequenceNodeInitTM Method

somfTSortedSequenceNodeInitT Method

Index

A

Abstract classes

- somf_TCollection 34
- somf_TIterator 174
- somf_TSequence 223
- somf_TSequenceliterator 236

C

Collection classes

- somf_MCollectible 1
- somf_MLinkable 8
- somf_MOrderableCollectible 14
- somf_TAssoc 21
- somf_TCollectibleLong 28
- somf_TCollection 34
- somf_TDeque 47
- somf_TDequeIterator 75
- somf_TDequeLinkable 83
- somf_TDictionary 89
- somf_TDictionaryIterator 132
- somf_THashTable 139
- somf_THashTableIterator 167
- somf_TIterator 174
- somf_TPrimitiveLinkedList 178
- somf_TPrimitiveLinkedListIterator 192
- somf_TPriorityQueue 199
- somf_TPriorityQueueIterator 217
- somf_TSequence 223
- somf_TSequenceliterator 236
- somf_TSet 243
- somf_TSetIterator 269
- somf_TSortedSequence 275
- somf_TSortedSequenceliterator 296
- somf_TSortedSequenceNode 307

D

Define

- SOMF_CALL_COMPARE_FN 1
- SOMF_CALL_HASH_FN 1
- SOMF_NIL 1

E

- EComparisonResult enum 14
- enum

- EComparisonResult 14

I

Iterator classes

- som_TDictionaryIterator 132
- somf_TDequeIterator 75
- somf_THashTableIterator 167
- somf_TPrimitiveLinkedListIterator 192
- somf_TPriorityQueueIterator 217
- somf_TSetIterator 269
- somf_TSortedSequenceliterator 296

M

Main collection classes

- somf_TDeque 47
- somf_TDictionary 89
- somf_THashTable 139
- somf_TPrimitiveLinkedList 178
- somf_TPriorityQueue 199
- somf_TSet 243
- somf_TSortedSequence 275

Mixin classes

- somf_MCollectible 1
- somf_MLinkable 8
- somf_MOrderableCollectible 14

Q

- Queues 47

S

- SOMF_CALL_BETTER_ORDERABLE_COMPARE_FN define 14
- SOMF_CALL_COMPARE_FN define 1
- SOMF_CALL_HASH_FN define 1
- SOMF_CALL_ORDERABLE_COMPARE_FN define 14
- somf_MBetterOrderableCompareFn typedef 14
- somf_MCollectible class 1
 - somfClone method 2
 - somfClonePointer method 3
 - somfHash method 4
 - somfIsEqual method 5
 - somfIsNotEqual method 6
 - somfIsSame method 7

somf_MCollectibleCompareFn typedef	1	somfAssign method	55
somf_MLinkable class	8	somfBefore method	56
somfGetNext method	9	somfCount method	57
somfGetPrevious method	10	somfCreateIterator method	58
somfMLinkableInit method	11	somfCreateNewLink method	59
somfSetNext method	12	somfCreateSequenceIterator method	60
somfSetPrevious method	13	somfDeleteAll method	61
somf_MollectibleHashFn typedef	1	somfFirst method	62
somf_MOrderableCollectible class	14	somfInsert method	63
somfCompare method	15	somfLast method	64
somfIsGreaterThan method	17	somfMember method	65
somfIsGreaterThanOrEqualTo method	18	somfPop method	66
somfIsLessThan method	19	somfPush method	67
somfIsLessThanOrEqualTo method	20	somfRemove method	68
somf_MOrderableCompareFn typedef	14	somfRemoveAll method	69
SOMF_NIL define	1	somfRemoveFirst method	70
somf_TAssoc class	21	somfRemoveLast method	71
somfGetKey method	22	somfRemoveQ method	72
somfGetValue method	23	somfTDequeueInitD method	73
somfSetKey method	24	somfTDequeueInitF method	74
somfSetValue method	25	somf_TDequeueIterator class	75
somfTAssocInitM method	26	somfFirst method	76
somfTAssocInitMM method	27	somfLast method	77
somf_TCollectibleLong class	28	somfNext method	78
somfGetValue method	29	somfPrevious method	80
somfHash method	30	somfRemove method	81
somfIsEqual method	31	somfTDequeueIteratorInit method	82
somfSetValue method	32	somf_TDequeLinkable class	83
somfTCollectibleLongInit method	33	somfGetValue method	84
somf_TCollection class	34	somfSetValue method	85
somfAdd method	35	somfTDequeLinkableInitDD method	86
somfAddAll method	36	somfTDequeLinkableInitDDM method	87
somfCount method	37	somf_TDictionary class	89
somfCreateIterator method	38	somfAdd method	91
somfDeleteAll method	39	somfAddKeyValuePairMM method	92
somfIsEqual method	40	somfAddKeyValuePairMMB method	94
somfMember method	41	somfAssign method	96
somfRemove method	42	somfCopyImplementation method	97
somfRemoveAll method	43	somfCount method	98
somfSetTestFunction method	44	somfCreateIterator method	99
somfTCollectionInit method	45	somfCreateNewImplementationF method	100
somfTestFunction method	46	somfCreateNewImplementationFL method	102
somf_TDeque class	47	somfCreateNewImplementationFLL method	104
somfAdd method	49	somfCreateNewImplementationFLLL method	106
somfAddAfter method	50	somfDeleteAll method	108
somfAddBefore method	51	somfDeleteAllKeys method	109
somfAddFirst method	52		
somfAddLast method	53		
somfAfter method	54		

- somfDeleteAllValues method 110
- somfDeleteKey method 111
- somfGetHashFunction method 113
- somfKeyAtM method 114
- somfKeyAtMF method 115
- somfMember method 117
- somfRemove method 118
- somfRemoveAll method 119
- somfSetHashFunction method 120
- somfTDictionaryInitD method 121
- somfTDictionaryInitF method 122
- somfTDictionaryInitFL method 123
- somfTDictionaryInitFLL method 125
- somfTDictionaryInitL method 127
- somfTDictionaryInitLL method 128
- somfTDictionaryInitLLF method 129
- somfValueAt method 131
- somf_TDictionaryIterator class 132
 - somfFirst method 133
 - somfNext method 135
 - somfRemove method 137
 - somfTDictionaryIteratorInit method 138
- somf_THashTable class 139
 - somfAddMM method 141
 - somfAddMMB method 142
 - somfAssign method 144
 - somfCount method 145
 - somfDelete method 146
 - somfDeleteAll method 147
 - somfDeleteAllKeys method 148
 - somfDeleteAllValues method 149
 - somfGetGrowthRate method 150
 - somfGetHashFunction method 151
 - somfGetRehashThreshold method 152
 - somfGrow method 153
 - somfMember method 154
 - somfRemove method 155
 - somfRemoveAll method 156
 - somfRetrieve method 157
 - somfSetGrowthRate method 158
 - somfSetHashFunction method 159
 - somfSetRehashThreshold method 160
 - somfTHashTableInitFL method 161
 - somfTHashTableInitFLL method 162
 - somfTHashTableInitFLLL method 164
 - somfTHashTableInitH method 166
- somf_THashTableIterator class 167
 - somfFirst method 168
 - somfHashTableIteratorInit method 173
 - somfNext method 170
 - somfRemove method 172
- somf_TIterator class 174
 - somfFirst method 175
 - somfNext method 176
 - somfRemove method 177
- somf_TPrimitiveLinkedList class 178
 - somfAddAfter method 179
 - somfAddBefore method 180
 - somfAddFirst method 180 to 181
 - somfAddLast method 182
 - somfAfter method 183
 - somfBefore method 184
 - somfCount method 185
 - somfFirst method 186
 - somfLast method 187
 - somfRemove method 188
 - somfRemoveAll method 189
 - somfRemoveFirst method 190
 - somfRemoveLast method 191
- somf_TPrimitiveLinkedListIterator class 192
 - somfFirst method 193
 - somfLast method 194
 - somfNext method 195
 - somfPrevious method 196
 - somfTPrimitiveLinkedListIteratorInit method 197
- somf_TPriorityQueue class 199
 - somfAdd method 201
 - somfAssign method 202
 - somfCount method 203
 - somfCreateIterator method 204
 - somfDeleteAll method 205
 - somfGetEqualityComparisonFunction method 206
 - somfInsert method 207
 - somfMember method 208
 - somfPeek method 209
 - somfPop method 210
 - somfRemove method 211
 - somfRemoveAll method 212
 - somfReplace method 213
 - somfSetEqualityComparisonFunction method 214
 - somfTPriorityQueueInitF method 215
 - somfTPriorityQueueInitP method 216
- somf_TPriorityQueueIterator class 217
 - somfFirst method 218
 - somfNext method 219

- somfRemove method 220
- somfTPriorityQueueIteratorInit method 221
- somf_TSequence class 223
 - somfAdd method 224
 - somfAfter method 225
 - somfBefore method 226
 - somfCount method 227
 - somfCreateIterator method 228
 - somfDeleteAll method 229
 - somfFirst method 230
 - somfLast method 231
 - somfOccurrencesOf method 232
 - somfRemove method 233
 - somfRemoveAll method 234
 - somfTSequenceInit method 235
- somf_TSequenceIterator class 236
 - somfFirst method 237
 - somfLast method 238
 - somfNext method 239
 - somfPrevious method 240
 - somfRemove method 241
- somf_TSet class 243
 - somfAdd method 245
 - somfAssign method 246
 - somfCount method 247
 - somfCreateIterator method 248
 - somfDeleteAll method 249
 - somfDifferenceS method 250
 - somfDifferenceSS method 251
 - somfGetHashFunction method 252
 - somfIntersectionS method 253
 - somfIntersectionSS method 254
 - somfMember method 255
 - somfRehash method 256
 - somfRemove method 257
 - somfRemoveAll method 258
 - somfSetHashFunction method 259
 - somfTSetInitF method 260
 - somfTSetInitFL method 261
 - somfTSetInitL method 262
 - somfTSetInitLF method 263
 - somfTSetInitS method 264
 - somfUnionS method 265
 - somfUnionSS method 266
 - somfXorS method 267
 - somfXorSS method 268
- somf_TSetIterator class 269
 - somfFirst method 270
 - somfNext method 271
- somfRemove method 273
- somfTSetIteratorInit method 274
- somf_TSortedSequence class 275
 - somfAdd method 277
 - somfAfter method 278
 - somfAssign method 279
 - somfBefore method 280
 - somfCount method 281
 - somfCreateIterator method 282
 - somfCreateSequenceIterator method 283
 - somfCreateSortedSequenceNode method 284
 - somfDeleteAll method 285
 - somfFirst method 286
 - somfGetSequencingFunction method 287
 - somfLast method 288
 - somfMember method 289
 - somfOccurrencesOf method 290
 - somfRemove method 291
 - somfRemoveAll method 292
 - somfSetSequencingFunction method 293
 - somfTSortedSequenceInitF method 294
 - somfTSortedSequenceInitS method 295
- somf_TSortedSequenceIterator class 296
 - somfFirst method 297
 - somfLast method 299
 - somfNext method 300
 - somfPrevious method 302
 - somfRemove method 303
 - somfStartHere method 304
- somf_TSortedSequenceIteratorInit method 305
- somf_TSortedSequenceNode class 307
 - somfGetKey method 308
 - somfGetLeftChild method 309
 - somfGetParent method 310
 - somfGetRed method 311
 - somfGetRightChild method 312
 - somfSetKey method 313
 - somfSetLeftChild method 314
 - somfSetParent method 315
 - somfSetRed method 316
 - somfSetRefOn method 317
 - somfSetRightChild method 318
 - somfTSortedSequenceNodeInitT method 319
 - somfTSortedSequenceNodeInitTM method 320
 - somfTSortedSequenceNodeInitTMT method 321
- somfAdd method 49, 91, 201, 224, 245, 277

somfAddAfter method 50, 179, 183
 somfAddAll method 36
 somfAddBefore method 51, 180
 somfAddFirst method 52, 180 to 181
 somfAddKeyValuePairMM method 92
 somfAddKeyValuePairMMB method 94
 somfAddLast method 53, 182
 somfAddMM method 141
 somfAddMMB method 142
 somfAfter method 54, 225, 278
 somfAssign method 55, 96, 144, 202, 246, 279
 somfBefore method 56, 184, 226, 280
 somfClone method 2
 somfClonePointer method 3
 somfCompare method 15
 somfCopyImplementation method 97
 somfCount method 57, 98, 145, 185, 203, 227, 247, 281
 somfCreateIterator method 58, 99, 204, 228, 248, 282
 somfCreateNewImplementationF method 100
 somfCreateNewImplementationFL method 102
 somfCreateNewImplementationFLL method 104
 somfCreateNewImplementationFLLL method 106
 somfCreateNewLink method 59
 somfCreateSequencelIterator method 60, 283
 somfCreateSortedSequenceNode method 284
 somfDelete method 146
 somfDeleteAll method 61, 108, 147, 205, 229, 249, 285
 somfDeleteAllKeys method 109, 148
 somfDeleteAllValues method 110, 149
 somfDeleteKey method 111
 somfDifferenceS method 250
 somfDifferenceSS method 251
 somfFirst method 62, 76, 133, 168, 175, 186, 193, 218, 230, 237, 270, 286, 297
 somfGetEqualityComparisonFunction method 206
 somfGetGrowthRate method 150
 somfGetHashFunction method 113, 151, 252
 somfGetKey method 22, 308
 somfGetLeftChild method 309
 somfGetNext method 9
 somfGetParent method 310
 somfGetPrevious method 10
 somfGetRed method 311
 somfGetRehashThreshold method 152
 somfGetRightChild method 312
 somfGetSequencingFunction method 287
 somfGetValue method 84
 somfGrow method 153
 somfHash method 4, 30
 somfInsert method 63, 207
 somfIntersectionS method 253
 somfIntersectionSS method 254
 somfIsEqual method 5, 31, 40
 somfIsGreaterThan method 17
 somfIsGreaterThanOrEqualTo method 18
 somfIsLessThan method 19
 somfIsLessThanOrEqualTo method 20
 somfIsNotEqual method 6
 somfIsSame method 7
 somfKeyAtM method 114
 somfKeyAtMF method 115
 somfLast method 64, 77, 187, 194, 231, 238, 288, 299
 somfMember method 65, 117, 154, 208, 255, 289
 somfMLinkableInit method 11
 somfNext method 78, 135, 170, 176, 195, 219, 239, 271, 300
 somfOccurrencesOf method 232
 somfOccurrencesOf method 290
 somfPeek method 209
 somfPop method 66, 210
 somfPrevious method 80, 196, 240, 302
 somfPush method 67
 somfRehash method 256
 somfRemove method 68, 81, 118, 137, 155, 172, 177, 188, 211, 220, 233, 241, 257, 273, 291, 303
 somfRemoveAll method 69, 119, 156, 189, 212, 234, 258, 292
 somfRemoveFirst method 70, 190
 somfRemoveLast method 71, 191
 somfRemoveQ method 72
 somfReplace method 213
 somfRetrieve method 157
 somfSetEqualityComparisonFunction method 214
 somfSetGrowthRate method 158
 somfSetHashFunction method 120, 159, 259
 somfSetKey method 24, 313
 somfSetLeftChild method 314
 somfSetNext method 12
 somfSetParent method 315
 somfSetPrevious method 13

- somfSetRed method 316
- somfSetRedOn method 317
- somfSetRehashThreshold method 160
- somfSetRightChild method 318
- somfSetSequencingFunction method 293
- somfSetTestFunction method 44
- somfSetValue method 85
- somfStartHere method 304
- somfTAssocInitM method 26
- somfTAssocInitMM method 27
- somfTCollectibleLongInit method 33
- somfTCollectionInit method 45
- somfTDequeueInitD method 73
- somfTDequeueInitF method 74
- somfTDequeueIteratorInit method 82
- somfTDequeueLinkableInitDD method 86
- somfTDequeueLinkableInitDDM method 87
- somfTDicctionaryInitD method 121
- somfTDicctionaryInitF method 122
- somfTDicctionaryInitFL method 123
- somfTDicctionaryInitFLL method 125
- somfTDicctionaryInitL method 127
- somfTDicctionaryInitLL method 128
- somfTDicctionaryInitLLF method 129
- somfTDicctionaryIteratorInit method 138
- somfTestFunction method 46
- somfTHashTableInitFL method 161
- somfTHashTableInitFLL method 162
- somfTHashTableInitFLLL method 164
- somfTHashTableInitH method 166
- somfTHashTableIteratorInit method 173
- somfTPrimitiveLinkedListIteratorInit method 197
- somfTPriorityQueueInitF method 215
- somfTPriorityQueueInitP method 216
- somfTPriorityQueueIteratorInit method 221
- somfTSequenceInit method 235
- somfTSetInitF method 260
- somfTSetInitFL method 261
- somfTSetInitL method 262
- somfTSetInitLF method 263
- somfTSetInitS method 264
- somfTSetIteratorInit method 274
- somfTSortedSequenceInitF method 294
- somfTSortedSequenceInitS method 295
- somfTSortedSequenceIteratorInit method 305
- somfTSortedSequenceNodeInitT method 319
- somfTSortedSequenceNodeInitTM method 320
- somfTSortedSequenceNodeInitTMT method 321

- somfUnionS method 265
- somfUnionSS method 266
- somfValueAt method 131
- somfXorS method 267
- somfXorSS method 268
- Stacks 47
- Supporting classes
 - somf_TAssoc 21
 - somf_TCollectibleLong 28
 - somf_TDequeueLinkable 83
 - somf_TSortedSequenceNode 307

T

- typedef
 - somf_MCollectibleCompareFn 1
 - somf_MCollectibleHashFn 1