SOMobjects Developer Toolkit

Collection Classes Reference Manual

Reference material for the classes and methods of the Collection Classes provided with the System Object Model

Version 2.1
October 1994

Version 2.1 (October 1994)

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THE PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR 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.

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

It is possible that this publication may contain reference 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 intends to announce such IBM products, programming, or services in your country.

Requests for technical information about IBM products should be made to your IBM Authorized Dealer or your IBM Marketing Representative.

IBM 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 OS/2 or AIX programming techniques. You may copy and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the OS/2 or AIX 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) (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.

The term "IBM" is a registered trademark and "SOMobjects" and "System Object Model" are trademarks of International Business Machines Corporation.

Notice to US Government Users – Documentation related to restricted rights – Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Collection Classes Reference Manual

Contents

Basics for Using the Collection Classes 1
Categories of Collection Classes 1
Abstract Classes 3
Main Collection Classes 4
Iterator Classes 6
Mixin Classes 7
Supporting Classes 8
Inheritance Hierarchy of the Collection Classes 9
Utility Collection Classes by Category 10

somf_MCollectible Class 11
somfClone Method 13
somfClonePointer Method 14
somfHash Method 15
somfIsEqual Method 16
somfIsNotEqual Method 17
somfIsSame Method 18

somf_MLinkable Class 19
somfGetNext Method 20
somfGetPrevious Method 21
somfMLinkableInit Method 22
somfSetNext Method 23
somfSetPrevious Method 24

somf_MOrderableCollectible Class 25
somfCompare Method 27
somfIsGreaterThan Method 29
somfIsGreaterThanOrEqualTo Method 30
somfIsLessThan Method 31
somfIsLessThanOrEqualTo Method 32

somf_TAssoc Class 33
somfGetKey Method 34
somfGetValue Method 35
somfSetKey Method 36
somfSetValue Method 37
somfTAssocInitM Method 38
somfTAssocInitMM Method 39

somf_TCollectibleLong Class 40
somfGetValue Method 41
somfHash Method 42
somfIsEqual Method 43
somfSetValue Method 44
somfTCollectibleLongInit Method 45

somf_TCollection Class 46
somfAdd Method 47
somfAddAll Method 48
somfCount Method 49
somfCreateIterator Method 50
somfDeleteAll Method 51
somfIsEqual Method 52
somfMember Method 53
somfRemove Method 54
somfRemoveAll Method 55
somfSetTestFunction Method 56
somfTCollectionInit Method 57
somfTestFunction Method 58

somf_TDeque Class 59
somfAdd Method 61
somfAddAfter Method 62
somfAddBefore Method 63
somfAddFirst Method 64
somfAddLast Method 65
somfAfter Method 66
somfAssign Method 67
somfBefore Method 68
somfCount Method 69
somfCreateIterator Method 70
somfCreateNewLink Method 71
somfCreateSequenceIterator Method 72
somfDeleteAll Method 73
somfFirst Method 74
somfInsert Method 75
somfLast Method 76
somfMember Method 77
somfPop Method 78
somfPush Method 79
somfRemove Method 80
somfRemoveAll Method 81
somfRemoveFirst Method 82
somfRemoveLast Method 83
somfRemoveQ Method 84
somfTDequeInitD Method 85
somfTDequeInitF Method 86

somf_TDequeIterator Class 87
somfFirst Method 88
somfLast Method 90
somfNext Method 91
somfPrevious Method 93
somfRemove Method 95
somfTDequeIteratorInit Method 97

somf_TDequeLinkable Class 98
somfGetValue Method 99
somfSetValue Method 100
somfTDequeLinkableInitDD Method 101
somfTDequeLinkableInitDDM Method 102

somf_TDictionary Class 103
somfAdd Method 105
somfAddKeyValuePairMM Method 107
somfAddKeyValuePairMMB Method 109
somfAssign Method 111
somfCopyImplementation Method 112
somfCount Method 113
somfCreateIterator Method 114
somfCreateNewImplementationF Method 115
somfCreateNewImplementationFL Method 117
somfCreateNewImplementationFLL Method 119
somfCreateNewImplementationFLLL Method 121
somfDeleteAll Method 123
somfDeleteAllKeys Method 125
somfDeleteAllValues Method 126
somfDeleteKey Method 127
somfGetHashFunction Method 128
somfKeyAtM Method 129
somfKeyAtMF Method 130
somfMember Method 132
somfRemove Method 133
somfRemoveAll Method 134
somfSetHashFunction Method 135
somfTDictionaryInitD Method 136
somfTDictionaryInitF Method 137
somfTDictionaryInitFL Method 138
somfTDictionaryInitFLL Method 139
somfTDictionaryInitL Method 141
somfTDictionaryInitLL Method 142
somfTDictionaryInitLLF Method 143
somfValueAt Method 145

somf_TDictionaryIterator Class 146
somfFirst Method 147
somfNext Method 149
somfRemove Method 151
somfTDictionaryIteratorInit Method 153

somf_THashTable Class 154
somfAddMM Method 156
somfAddMMB Method 158
somfAssign Method 160
somfCount Method 161
somfDelete Method 162
somfDeleteAll Method 164
somfDeleteAllKeys Method 166
somfDeleteAllValues Method 168
somfGetGrowthRate Method 170
somfGetHashFunction Method 171
somfGetRehashThreshold Method 172
somfGrow Method 173
somfMember Method 174

somfRemove Method 175
somfRemoveAll Method 176
somfRetrieve Method 177
somfSetGrowthRate Method 178
somfSetHashFunction Method 179
somfSetRehashThreshold Method 181
somfTHashTableInitFL Method 182
somfTHashTableInitFLL Method 183
somfTHashTableInitFLLL Method 184
somfTHashTableInitH Method 186

somf_THashTableIterator Class 187
somfFirst Method 188
somfNext Method 190
somfRemove Method 192
somfTHashTableIteratorInit Method 194

somf_TIterator Class 195
somfFirst Method 196
somfNext Method 197
somfRemove Method 198

somf_TPrimitiveLinkedList Class 199
somfAddAfter Method 200
somfAddBefore Method 201
somfAddFirst Method 202
somfAddLast Method 203
somfAfter Method 204
somfBefore Method 205
somfCount Method 206
somfFirst Method 207
somfLast Method 208
somfRemove Method 209
somfRemoveAll Method 210
somfRemoveFirst Method 211
somfRemoveLast Method 212

somf_TPrimitiveLinkedListIterator Class 213
somfFirst Method 214
somfLast Method 216
somfNext Method 217
somfPrevious Method 219
somfTPrimitiveLinkedListIteratorInit Method 220

somf_TPriorityQueue Class 221
somfAdd Method 223
somfAssign Method 224
somfCount Method 225
somfCreateIterator Method 226
somfDeleteAll Method 227
somfGetEqualityComparisonFunction Method 228
somfInsert Method 229
somfMember Method 230

somfPeek Method 231
somfPop Method 232
somfRemove Method 233
somfRemoveAll Method 234
somfReplace Method 235
somfSetEqualityComparisonFunction Method 236
somfTPriorityQueueInitF Method 237
somfTPriorityQueueInitP Method 238

somf_TPriorityQueueIterator Class 239
somfFirst Method 240
somfNext Method 242
somfRemove Method 244
somfTPriorityQueueIteratorInit Method 245

somf_TSequence Class 246
somfAdd Method 247
somfAfter Method 248
somfBefore Method 249
somfCount Method 250
somfCreateIterator Method 251
somfDeleteAll Method 252
somfFirst Method 253
somfLast Method 254
somfOccurrencesOf Method 255
somfRemove Method 256
somfRemoveAll Method 257
somfTSequenceInit Method 258

somf_TSequenceIterator Class 259
somfFirst Method 260
somfLast Method 261
somfNext Method 262
somfPrevious Method 263
somfRemove Method 264

somf_TSet Class 265
somfAdd Method 267
somfAssign Method 268
somfCount Method 269
somfCreateIterator Method 270
somfDeleteAll Method 271
somfDifferenceS Method 272
somfDifferenceSS Method 273
somfGetHashFunction Method 274
somfIntersectionS Method 275
somfIntersectionSS Method 276
somfMember Method 277
somfRehash Method 278
somfRemove Method 279
somfRemoveAll Method 280
somfSetHashFunction Method 281
somfTSetInitF Method 282

somfTSetInitFL Method 283
somfTSetInitL Method 284
somfTSetInitLF Method 285
somfTSetInitS Method 286
somfUnionS Method 287
somfUnionSS Method 288
somfXorS Method 289
somfXorSS Method 290

somf_TSetIterator Class 291
somfFirst Method 292
somfNext Method 294
somfRemove Method 296
somfTSetIteratorInit Method 298

somf_TSortedSequence Class 299
somfAdd Method 301
somfAfter Method 302
somfAssign Method 303
somfBefore Method 304
somfCount Method 305
somfCreateIterator Method 306
somfCreateSequenceIterator Method 307
somfCreateSortedSequenceNode Method 308
somfDeleteAll Method 309
somfFirst Method 310
somfGetSequencingFunction Method 311
somfLast Method 312
somfMember Method 313
somfOccurrencesOf Method 315
somfRemove Method 316
somfRemoveAll Method 317
somfSetSequencingFunction Method 318
somfTSortedSequenceInitF Method 319
somfTSortedSequenceInitS Method 320

somf_TSortedSequenceIterator Class 321
somfFirst Method 322
somfLast Method 324
somfNext Method 325
somfPrevious Method 327
somfRemove Method 329
somfStartHere Method 331
somfTSortedSequenceIteratorInit Method 333

somf_TSortedSequenceNode Class 334
somfGetKey Method 335
somfGetLeftChild Method 336
somfGetParent Method 337
somfGetRed Method 338
somfGetRightChild Method 339
somfSetKey Method 340
somfSetLeftChild Method 341

somfSetParent Method 342
somfSetRed Method 343
somfSetRedOn Method 344
somfSetRightChild Method 345
somfTSortedSequenceNodeInitT Method 346
somfTSortedSequenceNodeInitTM Method 347
somfTSortedSequenceNodeInitTMT Method 348

About This Book

This book gives reference material for the **Collection Classes** provided with the **SOMobjects Developer Toolkit.** In particular, it contains a reference page for every class included in the Collection Classes, and for each of their methods.

In addition to this book, refer to the *SOMobjects Developer Toolkit: Programmers Reference Manual* for information about the other classes, methods, functions, and macros provided in the SOMobjects Toolkit, and to the *SOMobjects Developer Toolkit Users Guide* for introductory information. Also, refer to the *Emitter Framework Guide and Reference* for documentation of the Emitter Framework of the SOMobjects Developer Toolkit.

How This Book Is Organized

The first part of this book contains some general information about the Collection Classes and the categories by which they are organized.The reference pages in this book describe the classes in alphabetical order, with the methods of each class given in alphabetical order following their corresponding class.

The reference page for a **class** contains the following topics:

Description:	A description of the class.
File Stem:	The file stem for the class's IDL interface specification (.idl) file and its usage binding (.h/.xh) files.
Base:	The class's direct base (parent) classes.
Ancestor Classes:	The class's ancestor (indirect base) classes.
Metaclass:	The class's metaclass.
New Methods:	The names of the methods that the class introduces (grouped roughly according to purpose). Each new method is documented on a separate reference page.
Overriding Methods:	The names of the methods that the class overrides from ancestor classes

The reference page for a **method** contains the following topics:

Purpose:	The purpose of the method in brief.
Syntax:	The method's C/C++ procedure prototype (which includes the method procedure's return type and the names and types of its parameters). The in/out/inout keywords associated with each of the method's parameters in the method's IDL declaration are also shown. These keywords are shown for information only; they are not actually present in the method procedure prototype.
Description:	A description of the method's use.
Parameters:	A description of each of the method procedure's parameters**.**
Return Value:	A description of the method's return value.
Example:	An example of using or overriding the method, if available. Although methods of SOM classes are language neutral (i.e., they can be invoked from any programming language that can use the SOMobjects system), the examples given here are written in C.
Original Class:	The name of the class that introduces the method (the class is documented separately in this book)**.**
Related Information:	Related methods that can be found in this book.

Who Should Use This Book

This book is for the professional programmer using the SOMobjects Toolkit to build object-oriented class libraries or application programs that use SOM class libraries or the frameworks in the SOMobjects Toolkit.

This book assumes that you are an experienced programmer and that you have a general familiarity with the basic notions of object-oriented programming. Practical experience using an object-oriented programming language is helpful, but not essential.

Collection Classes Reference Manual

Basics for Using the Collection Classes

The Collection Classes constitute a large group of classes and methods provided for the programmer's convenience. Collection Classes — sometimes also called Foundation Classes — are a set of classes whose purpose is to contain other objects. These classes and their related methods implement most of the common data structures encountered in programming, thus relieving the programmer of those coding tasks. The collection classes can be used in client code "as is," or they can be used as the basis for deriving new classes.

Categories of Collection Classes

The collection classes are organized into the following categories:

- **Abstract classes** — Define the conceptual operations that are implemented by methods in other (sub)classes.
- **Main collection classes** — Represent each of the implemented data structures for collecting elements into a group.
- **Iterator classes** — Define an iterator class corresponding to each of the main collection classes, enabling clients to iterate through each of the objects in the collection.
- **Mixin classes** — Define characteristics that apply to more than one kind of collection class, such as ordering or linking. A collection class may also require certain mixin characteristics in objects that it collects. To facilitate this, a mixin class can be "mixed in" with an existing user class to derive a new "collectible" class.
- **Supporting classes** — Provides additional capabilities used internally by collection classes; is of interest primarily to those deriving new collection classes.

Each group of the foregoing classes is discussed more thoroughly in subsequent topics, with particular emphasis on the main collection classes.

IsSame vs. IsEqual comparisons

The distinction between the **IsSame** vs. **IsEqual** operation is an important concept when making comparisons. The various collection classes use one or the other of these approaches to compare contained objects. The operations are defined as follows:

1. **IsSame** is true when two objects are really the *same* object. This means "both" objects are literally the same object; that is, both parts of a comparison are testing the same instantiation.
2. **IsEqual** is true when two objects are *equivalent* objects. This would occur when two different instantiations contain the same values, or at least values which, for the sake of the comparison, can be considered the same. Stated differently, the two instantiations are isomorphic.

Class inheritance vs. element inheritance

There are two distinct aspects of inheritance that pertain to each class:

1. The inheritance of the *collection class* itself that is derived from its parent or base class.
2. The inheritance of the *elements* or objects that can be inserted into a collection class as a value.

Do not assume that these two inheritances are the same. There are times when a collection class has one parent, but the objects to be inserted into the collection class may have a totally different parent. Further, a collection class may mandate that elements (or values) meet certain inheritance requirements before the elements can be stored in that collection container.

The subsequent topics describing each collection class also discuss any pertinent inheritance restrictions of the class and its contained elements.

Object-initializer methods

Most of the collection classes provide optional initializers. These are methods with which a newly created instance can be initialized to some state other than its default. All initializer methods use the following format:

somf<*className*>Init<*optional postfix to distinguish between initializers*>

The initializers can also be used to reset certain default properties of the collection classes. For example, although the **somf_THashTable** class uses an **IsSame** approach for comparing objects, the initializer method could be used to cause an instance of **somf_THashTable** to compare using **IsEqual** instead.

Some initializer methods cannot be overridden by inheriting classes. That is, the initializer methods can be used, but they cannot be redefined.

Naming conventions

The class names for Mixin classes all begin with the prefix **somf_M.** All other collection classes have names beginning with the prefix **somf_T.**

The method names of methods defined by the collection classes all begin with the prefix **somf**, without an underscore after the prefix.

Abstract Classes

The Annotated C++ Reference Manual[1] describes an abstract class as follows: "The abstract class mechanism supports the notion of a general concept, such as a `shape`, of which only more concrete variants, such as `circle` and `square`, can actually be used. An abstract class can also be used to define an interface for which derived classes provide a variety of implementations."

The concept of an *abstract base class*, which was briefly discussed earlier in this manual, is a C++ variation on the *abstract class*. An abstract base class is a class that not only describes the general concept, it also can not be instantiated. Another aspect of the abstract base class is the notion of *pure virtual function*. Any child of the parent abstract base class *must* override each pure virtual function (method) in order to use the function.

While the idea of a pure virtual function is primarily a C++ concept, it is a valid concept in SOM as well. This is especially true when defining basic behavior in a parent class that applies for all children of that class. This concept allows class implementors the flexibility to use either of two approaches:

1. Declare an interface in the parent class to a method that all children *must* override and redefine. If the method is *not* overridden, the parent class will print a corresponding message and processing will halt.
2. Declare and define an interface in the parent to a method that the children can either accept as their base definition or can override and redefine.

The **somf_TIterator** class provides an example of an abstract class which declares a method interface in the parent that all children *must* override and redefine. Specifically, the **somf_TIterator** class declares the methods **somfFirst** and **somfNext,** which all children derived from **somf_TIterator** must override.

The Abstract Classes include the following classes:

somf_TCollection — Represents a group of objects.

somf_TIterator — Declares the behavior common to all iterator classes. An iterator for a particular collection class will iterate over each element contained in an object of that class.

somf_TSequence — Declares the behavior common to all collections whose elements are ordered.

somf_TSequenceIterator — Declares the behavior of all iterators for children of the **somf_TSequence** class. This class is also a child of the **somf_TIterator** class.

1. Margaret A. Ellis and Bjarne Stroustrup, *The Annotated C++ Reference Manual* (Addison–Wesley Publishing Company, 1992).

Main Collection Classes

The set of main collection classes contains data structure classes for the following kinds of data structures, as described in the subsequent classes:

somf_THashTable — A table consisting of (*key, value*) pairs. The “key” provides the means for mapping into the table, and the “value” is the data element to be stored in the hash table.

somf_TDictionary — An unordered data structure with (key, value) pairs.

somf_TSet — An unordered collection of objects where the objects can only appear once.

somf_TDeque — A queue, stack, or deque collection.

somf_TPrimitiveLinkedList — A collection where each element in the list is linked to the element in front of it and also to the element behind it.

somf_TSortedSequence — A collection where the order of its elements is determined by how those elements relate to each other.

somf_TPriorityQueue — A special case of the sorted sequence.

Choosing the best class

If you are unsure which main collection class you should use, the following selection chart may prove helpful.

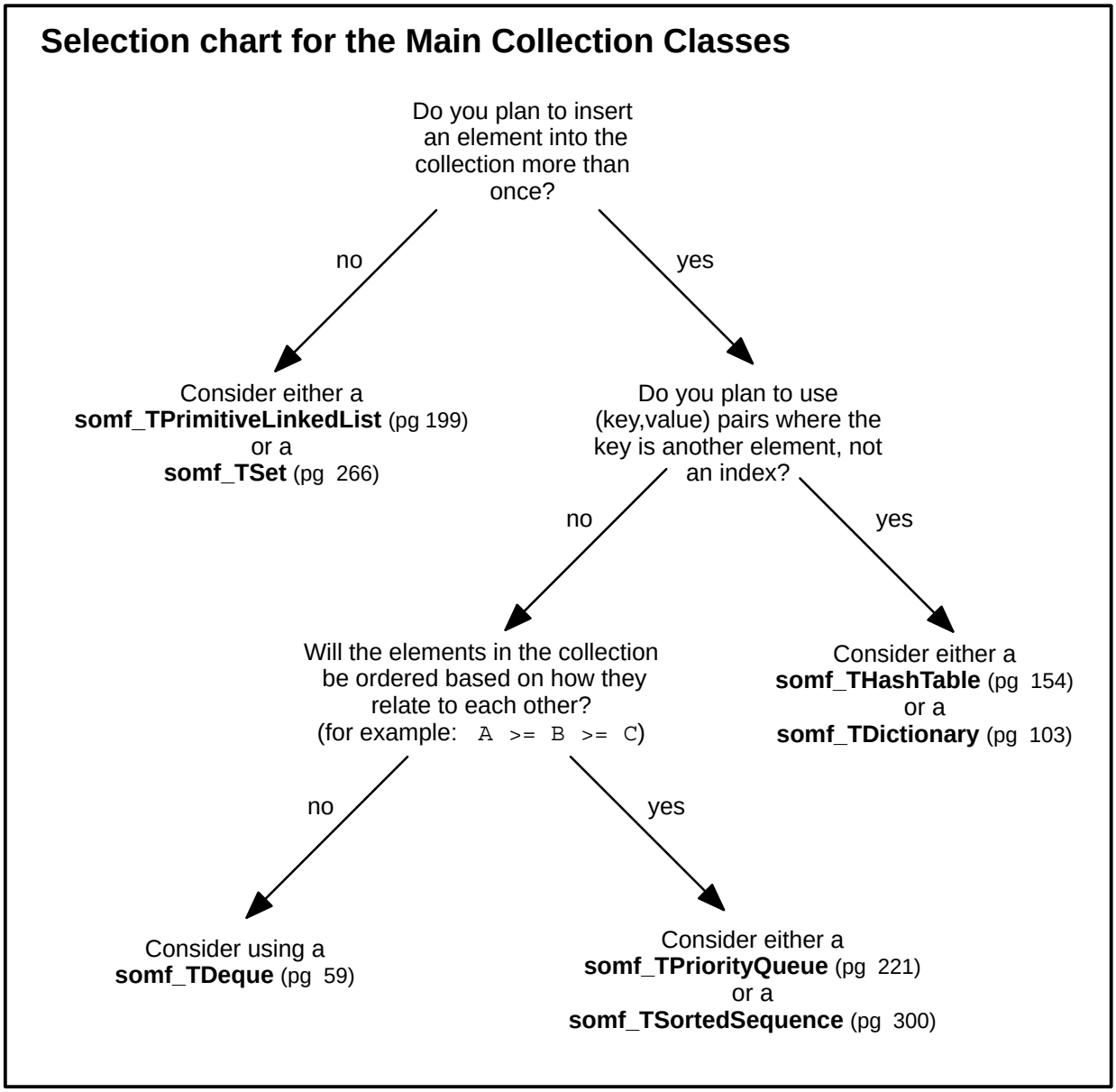

Figure 1. Collection class selection chart

Iterator Classes

Each of the main collection classes has a corresponding iterator class defined for it. An iterator for a particular object will iterate over each contained object in the collection. To illustrate, the following example uses an iterator of class **somf_TSetIterator** to iterate over each element in an object of the **somf_TSet** class.

```
somf_TSet set;
somf_TSetIterator itr;
somf_MCollectible obj;
Environment *ev;

set = somf_TSetNew();
ev = somGetGlobalEnvironment();

/* A bunch of stuff happens to set */

itr = somf_TSet_somfCreateIterator(set,ev);
obj = somf_TSetIterator_somfFirst(itr,ev);
while (obj != SOMF_NIL)
{
        /* do something to obj */
        obj = somf_TSetIterator_somfNext(itr,ev);
}
```

The **somfFirst** method is used to get the first element in the collection, and the **somfNext** method is used thereafter to get each “next” element. Using an iterator allows you to sequentially look at each element in the collection and do some appropriate processing on it.

Notice that the iterator was initialized using the method **somfCreateIterator**. All classes that inherit from **somf_TCollection** must provide a **somfCreateIterator** method. This shows one of the two ways to initialize an iterator; the other way is to use the constructor-initializer associated with the iterator. For example, `itr` could have been declared using:

```
somf_TSetIterator *itr;
itr = somf__TSetIteratorNew();
somf_TSetIterator_somfTSetIteratorInit(itr, ev, set);
```

Note: Some people may wonder why the iterator logic was not included in the main collection classes, rather than being in a separate class. One reason was so that the user can create multiple iterators for a single instance of a collection class. If the methods were part of the main collection classes, each instance would be limited to the one iterator that came with it.

If a collection changes while the iterator is in use, the iterator becomes invalid and will issue a notice that it cannot continue to the next element. So, for example, if a client program calls the collection’s **somfAdd** method after starting to iterate through the collection, the iterator will not allow processing to continue. The iterator will have to be reset, and the easiest way to do that is to call the iterator’s **somfFirst** method and start over.

If a collection is ordered, the iterator returns its elements in the correct order. If the collection is unordered or partially ordered (like **somf_TPriorityQueue**), the iterator returns its elements in some random order.

Mixin Classes

A Mixin class is a class designed "to be mixed in together with other classes to produce new subclasses."[2] Mixin classes do not necessarily describe stand-alone characteristics — just characteristics that may be common to more than one kind of class. For example, classes describing a "car" or a "dress" might both inherit from a Mixin class describing "red."

Another characteristic of mixin classes is that they inherit *only* from other mixin classes (not including **SOMObject**). A mixin class can *not* inherit from some other base class without a **somf_M** prefix.

For any object to be eligible for insertion into one of the main collection classes, that object *must* inherit from a Mixin class (see the table below). This is necessary because the mixin class declares certain behavior that the main collection class requires in the object in order to process it. For example, the **somf_MCollectible** mixin class declares the **somfIsEqual** method that is needed to compare objects in almost every collection.

To utilize a collection class for storing their own objects, programmers must first define a class whose instances will contain the required mixin characteristics. By using multiple inheritance, a class can inherit from zero or more mixin classes, as well as from its logical parent (if it has a logical parent). For example, if you want to store objects in a sorted sequence structure of class **somf_TSortedSequence**, those objects must be instances of a class defined to inherit characteristics from the **somf_MOrderableCollectible** mixin class, such as:

```
interface MySortSeqData : MyData, somf_MOrderableCollectible
```

There are three important mixin classes used by the main collection classes:

somf_MCollectible — Defines the generic methods needed by objects inserted into *any* of the collections classes. It provides the profile for the methods **somfIsEqual**, **somfIsSame**, and **somfHash**.

somf_MLinkable — Defines the general characteristics of objects that contain links.

somf_MOrderableCollectible — Defines the general characteristics of objects that are ordered.

The following table maps each main collection class to the mixin class from which an object must inherit in order for that object to be eligible for insertion into the corresponding main collection class:

Main Collection Class	Mixin Class from which an inserted object must inherit
somf_TDeque	**somf_MCollectible**
somf_TDictionary	**somf_MCollectible**
somf_THashTable	**somf_MCollectible**
somf_TPrimitiveLinkedList	**somf_MLinkable**
somf_TPriorityQueue	**somf_MOrderableCollectible**
somf_TSet	**somf_MCollectible**
somf_TSortedSequence	**somf_MOrderableCollectible**

2. Grady Booch, *Object Oriented Design with Applications* (Redwood City, California: The Benjamin/ Cummings Publishing Company, 1991), pg. 58

Supporting Classes

Many of the main collection classes use supporting classes. The **somf_TSortedSequence** class, for example, uses the supporting class **somf_TSortedSequenceNode** to define the behavior of a single node in a sorted sequence collection.

Included are the following supporting classes:

somf_TAssoc — Is used to hold a pair of objects.

somf_TDequeLinkable — Inherits from **somf_MLinkable** and provides a generic version of **somf_MLinkable** containing a long value. The **somf_TDequeLinkable** class is used by **somf_TDeque**.

somf_TSortedSequenceNode — Represents a node in a tree containing elements of the **somf_MOrderableCollectible** class. It contains a key (the **somf_MOrderableCollectible**) and a link to a left child and a right child.

somf_TCollectibleLong — Provides a generic **somf_MCollectible** class containing a long value.

The **somf_TDequeLinkable** and **somf_TSortedSequenceNode** classes will probably not be of particular interest unless you plan to derive a new collection class. **somf_TAssoc** may only be of interest if you are working with **somf_THashTable** or **somf_TDictionary**, since these two classes store key, value pairs. **somf_TCollectibleLong** will be of interest if you need a generic **somf_MCollectible** containing a long. **somf_TCollectibleLong** is not used by any of the other Collection Classes.

Inheritance Hierarchy of the Collection Classes

The inheritance hierarchy for the collection classes is depicted in the following chart. Note that this diagram does *not* illustrate all of the classes, only those which have some position in the inheritance hierarchy of the set.

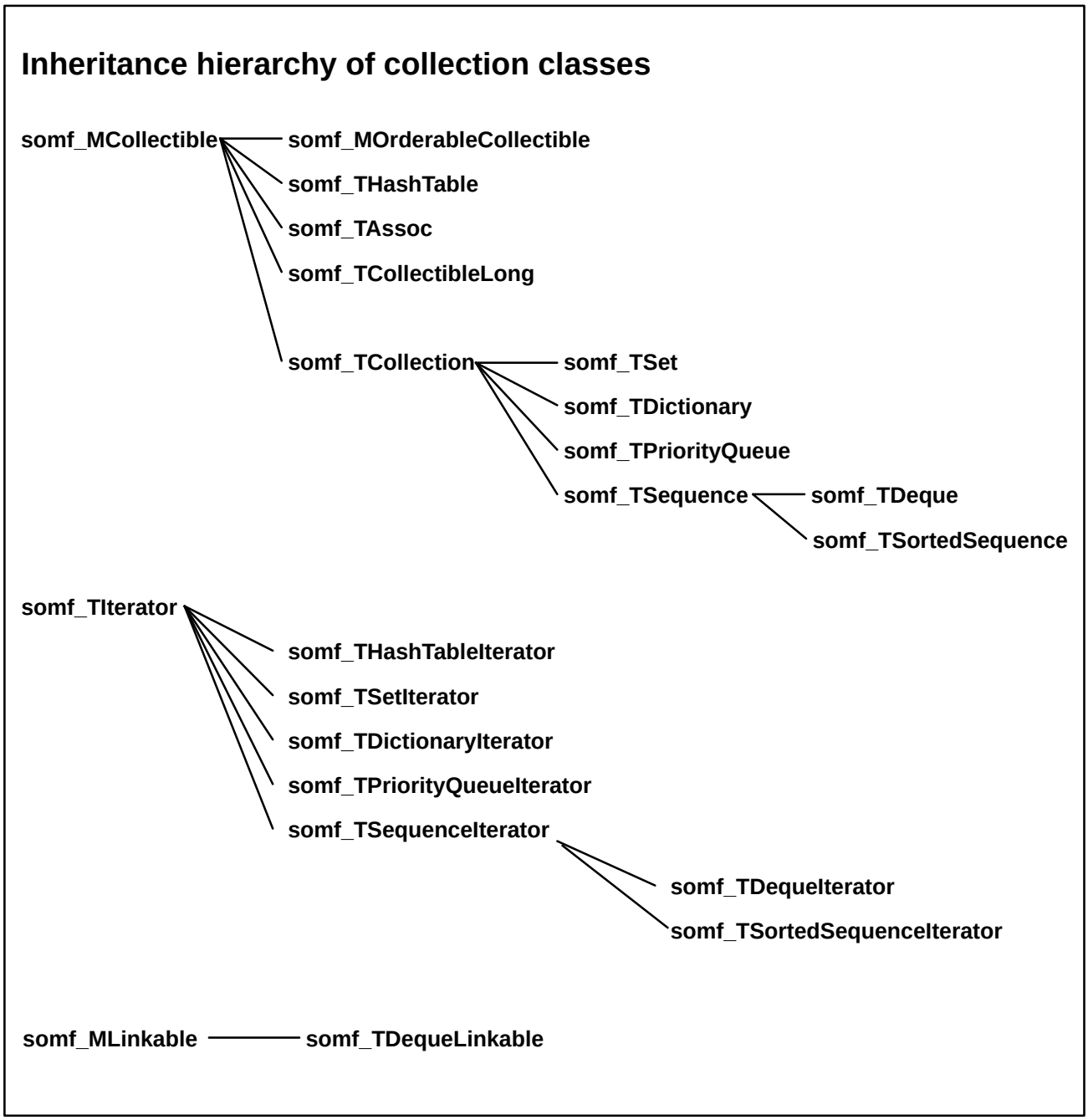

Figure 2. Collection classes inheritance hierarchy

Utility Collection Classes by Category

Following is the entire list of utility collection classes.

Abstract Classes

somf_TCollection
somf_TIterator
somf_TSequence
somf_TSequenceIterator

Main Collection Classes

somf_TDeque
somf_TDictionary
somf_THashTable
somf_TPrimitiveLinkedList
somf_TPriorityQueue
somf_TSet
somf_TSortedSequence

Iterator Classes

somf_TDequeIterator
somf_TDictionaryIterator
somf_THashTableIterator
somf_TPrimitiveLinkedListIterator
somf_TPriorityQueueIterator
somf_TSetIterator
somf_TSortedSequenceIterator

Mixin Classes

somf_MCollectible
somf_MLinkable
somf_MOrderableCollectible

Supporting Classes

somf_TAssoc
somf_TCollectibleLong
somf_TDequeLinkable
somf_TSortedSequenceNode

somf_MCollectible Class

Description

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 primary reason new classes inherit from **somf_MCollectible** is so that instances of the inheriting class can be inserted into one of the main collection classes (that is, into a **somf_THashTable**, **somf_TSet**, **somf_TDictionary**, or whatever).

All classes that inherit from **somf_MCollectible** *must* override either method **somfIsEqual** or **somfIsSame**, depending on which method the new class plans to use for comparison. The **somfHash** method will probably need to be overridden too.

This class is not thread-safe.

File Stem

mcollect

Base

SOMObject

Metaclass

SOMClass

Ancestor Classes

SOMObject

New Methods

somfClone
somfClonePointer
somfHash
somfIsEqual
somfIsSame
somfIsNotEqual

Overriding Methods

None.

Typedefs

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

somf_MCollectibleCompareFn
A method pointer to a **somfIsEqual or somfIsSame** 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

Purpose

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

Related Information

Methods: somfClonePointer

somfClonePointer Method

Purpose

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

Related Information

Methods: somfClone

somfHash Method

Purpose

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,** 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

somfIsEqual Method

Purpose

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

IDL Syntax

boolean somfIsEqual (in somf_MCollectible *obj*);

Description

The **somfIsEqual** 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, etc. The choice is to use either the **somfIsEqual** method or else the **somfIsSame** 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 (_somfIsEqual(obj, ev, obj2))
   somPrintf(" obj is equal to obj2\n");

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

Original Class

somf_MCollectible

Related Information

Methods: somfIsNotEqual

somfIsNotEqual Method

Purpose

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

IDL Syntax

boolean somfIsNotEqual (in somf_MCollectible *obj*);

Description

The **somfIsNotEqual** method returns `TRUE` if the specified object `obj` is not isomorphic to the receiving object.

This method uses the **somfIsEqual** method described on page 16. If a class inherits from **somf_MCollectible**, **somfIsEqual** *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 (_somfIsNotEqual(obj, ev, obj2))
   somPrintf(" obj is NOT equal to obj2\n");

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

Original Class

somf_MCollectible

Related Information

Methods: somfIsEqual

somfIsSame Method

Purpose

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

IDL Syntax

boolean somfIsSame (in somf_MCollectible *obj*);

Description

The **somfIsSame** 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 (_somfIsSame(obj, ev, obj2))
   somPrintf(" obj is the same as obj2\n");

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

Original Class

somf_MCollectible

somf_MLinkable Class

Description

This class defines the general characteristics of objects that contain links. For example, **somf_TPrimitiveLinkedList** 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

Metaclass

SOMClass

Ancestor Classes

SOMObject

New Methods

somfGetNext
somfSetNext
somfGetPrevious
somfSetPrevious
somfMLinkableInit

Overriding Methods

somInit

somfGetNext Method

Purpose

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

Related Information

Methods: somfSetNext, somfSetPrevious

somfGetPrevious Method

Purpose

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

Related Information

Methods: somfSetPrevious, somfGetNext

somfMLinkableInit Method

Purpose

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

IDL Syntax

somf_MLinkable somfMLinkableInit (
in somf_MLinkable *n*,
in somf_MLinkable *p*);

Description

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

Note: You cannot override this method.

Parameters

receiver	A pointer to an object of class **somf_MLinkable**.
ev	A pointer to the **Environment** structure for the calling method.
n	A pointer to the next **somf_MLinkable** object.
p	A pointer to the previous **somf_MLinkable** object.

Return Value

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

Example

```
somf_MLinkable ml;
Environment *ev;

ev = somGetGlobalEnvironment();

ml = somf_MLinkableNew();
_somfMLinkableInit(ml, ev, SOMF_NIL, SOMF_NIL);

_somFree (ml);
```

Original Class

somf_MLinkable

somfSetNext Method

Purpose

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.

Return Value

None.

Example

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

ev = somGetGlobalEnvironment();

ml = somf_MLinkableNew();
ml2 = somf_MLinkableNew();

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

/* Set ml2 to point to ml as the next link */
_somfSetNext(ml2, ev, ml);

_somFree (ml);
_somFree (ml2);
```

Original Class

somf_MLinkable

Related Information

Methods: somfSetPrevious, somfGetNext

somfSetPrevious Method

Purpose

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.

Return Value

None.

Example

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

ev = somGetGlobalEnvironment();

ml = somf_MLinkableNew();
ml2 = somf_MLinkableNew();

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

/* Set ml2 to point to ml as the previous link */
_somfSetPrevious(ml2, ev, ml);

_somFree (ml);
_somFree (ml2);
```

Original Class

somf_MLinkable

Related Information

Methods: somfSetNext, somfGetPrevious

somf_MOrderableCollectible Class

Description

Characteristics of the **somf_MOrderableCollectible** class should be mixed into objects that might need to be ordered. Objects passed to an instance of class **somf_TPriorityQueue** or **somf_TSortedSequence** (or their children) *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, such as an element in a **somf_TPriorityQueue** or **somf_TSortedSequence.**

All classes that inherit from **somf_MOrderableCollectible** *must* override the **somfIsEqual** method that is inherited from **somf_MCollectible**, as well as **somf_MOrderableCollectible**'s **somfIsLessThan** and **somfIsGreaterThan** methods.

This class is not thread-safe.

File Stem

morder

Base

somf_MCollectible

Metaclass

SOMClass

Ancestor Classes

somf_MCollectible, SOMObject

New Methods

somfIsGreaterThan
somfIsLessThan
somfCompare
somfIsGreaterThanOrEqualTo
somfIsLessThanOrEqualTo

Overriding Methods

None

Typedefs

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

somf_MOrderableCompareFn
A method pointer to a **somfIsLessThan** or **somfIsGreaterThan** 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

Purpose

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**, as well as the **somfIsLessThan** and **somfIsGreaterThan** methods of **somf_MOrderableCollectible,** *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);
```

Original Class

somf_MOrderableCollectible

Related Information

Methods: somfIsEqual, somfIsGreaterThan, somfIsLessThan

somfIsGreaterThan Method

Purpose

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

IDL Syntax

boolean somfIsGreaterThan (in somf_MOrderableCollectible *obj*);

Description

The **somfIsGreaterThan** method returns TRUE if the specified object `obj` is “greater than” the receiving object.

This method *must* be overridden if a class inherits from **somf_MOrderableCollectible**. If it is not overridden, 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 Greater Than” 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 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 ( _somfIsGreaterThan(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

Related Information

Methods: somfIsGreaterThanOrEqualTo, somfIsLessThan, somfIsEqual

somfIsGreaterThanOrEqualTo Method

Purpose

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

IDL Syntax

boolean somfIsGreaterThanOrEqualTo (in somf_MOrderableCollectible *obj*);

Description

The **somfIsGreaterThanOrEqualTo** 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 ( _somfIsGreaterThanOrEqualTo(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

Related Information

Methods: somfIsGreaterThan, somfIsLessThan, somfIsEqual

somfIsLessThan Method

Purpose

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

IDL Syntax

boolean somfIsLessThan (in somf_MOrderableCollectible *obj***);**

Description

The **somfIsLessThan** method returns TRUE if the specified object `obj` is “less than” the receiving object.

This method *must* be overridden if a class inherits from **somf_MOrderableCollectible**. If it is not overridden, 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. If you invoke this method directly, an error message is written and processing will end. 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 ( _somfIsLessThan(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

Related Information

Methods: somfIsLessThanOrEqualTo, somfIsEqual, somfIsGreaterThan

somfIsLessThanOrEqualTo Method

Purpose

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

IDL Syntax

boolean somfIsLessThanOrEqualTo (in somf_MOrderableCollectible *obj*);

Description

The **somfIsLessThanOrEqualTo** 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 ( _somfIsLessThanOrEqualTo(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

Related Information

Methods: somfIsLessThan, somfIsGreaterThan, somfIsEqual

somf_TAssoc Class

Description

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

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

somf_MCollectible

Metaclass

SOMClass

Ancestor Classes

somf_MCollectible, SOMObject

New Methods

somfGetKey
somfGetValue
somfSetKey
somfSetValue
somfTAssocInitM
somfTAssocInitMM

Overriding Methods

somInit
somUninit

somfGetKey Method

Purpose

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

IDL Syntax

somf_MCollectible somfGetKey ( );

Description

The **somfGetKey** method obtains the key of the associated (key, value) 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

Related Information

Methods: somfSetKey, somfSetValue, somfGetValue

somfGetValue Method

Purpose

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

IDL Syntax

somf_MCollectible somfGetValue ( );

Description

The **somfGetValue** method gets the value to the associated (key, value) 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

Related Information

Methods: somfSetValue, somfSetKey, somfGetKey

somfSetKey Method

Purpose

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

IDL Syntax

void somfSetKey (in somf_MCollectible *k*);

Description

The **somfSetKey** method sets the key of an associated (key, value) 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** which will be the key of the associated pair.

Return Value

None.

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

Related Information

Methods: somfGetKey, somfSetValue, somfGetValue

somfSetValue Method

Purpose

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 (key, value) 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** which will be the value of the associated pair.

Return Value

None.

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

Related Information

Methods: somfGetValue, somfGetKey, somfSetKey

somfTAssocInitM Method

Purpose

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 (key, value) 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** 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

Related Information

Methods: somfTAssocInitMM

somfTAssocInitMM Method

Purpose

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 (key, value) 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** 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

Related Information

Methods: somfTAssocInitM

somf_TCollectibleLong Class

Description

This class provides the user with a generic **somf_MCollectible** class 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

Metaclass

SOMClass

Ancestor Classes

somf_MCollectible, SOMObject

New Methods

somfGetValue
somfSetValue
somfTCollectibleLongInit

Overriding Methods

somInit
somfIsEqual
somfHash

somfGetValue Method

Purpose

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

Related Information

Methods: somfSetValue, somfIsEqual

somfHash Method

Purpose

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 (overridden here)

Related Information

Methods: somfGetValue, somfSetValue, somfTCollectibleLongInit, somfIsEqual

somfIsEqual Method

Purpose

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

IDL Syntax

boolean somfIsEqual (in somf_MCollectible *obj*);

Description

The **somfIsEqual** 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, etc.

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** 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

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

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

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

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

Original Class

somf_MCollectible (overridden here)

Related Information

Methods: somfGetValue, somfSetValue, somfTCollectibleLongInit

somfSetValue Method

Purpose

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.

Return Value

None.

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

Related Information

Methods: somfGetValue, somfTCollectibleLongInit, somfIsEqual

somfTCollectibleLongInit Method

Purpose

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

Related Information

Methods: somfSetValue, somfGetValue, somfIsEqual, somfHash

somf_TCollection Class

Description

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

File Stem

tcollect

Base

somf_MCollectible

Metaclass

SOMClass

Ancestor Classes

somf_MCollectible, SOMObject

New Methods

somfAdd
somfAddAll
somfRemove
somfRemoveAll
somfDeleteAll
somfCount
somfMember
somfCreateIterator
somfTestFunction
somfSetTestFunction
somfTCollectionInit

Overriding Methods

somfIsEqual

somfAdd Method

Purpose

Adds a specified `obj` to a collection.

IDL Syntax

somf_MCollectible somfAdd (in somf_MCollectible *obj*);

Description

The **somfAdd** 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 **somf_TCollection.**

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

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

Return Value

Two valid return values are possible for this method:

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

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

Example

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

Original Class

somf_TCollection

Related Information

Methods: somfAddAll

somfAddAll Method

Purpose

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.

Return Value

None.

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

Related Information

Methods: somfAdd

somfCount Method

Purpose

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 (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 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** or **somf_TDictionary** or any of the other classes that inherit from **somf_TCollection**.

Original Class

somf_TCollection

somfCreateIterator Method

Purpose

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

IDL Syntax

somf_TIterator somfCreateIterator ( );

Description

The **somfCreateIterator** 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** or **somf_TDictionary** or any of the other classes that inherit from **somf_TCollection**.

Original Class

somf_TCollection

somfDeleteAll Method

Purpose

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 (that is, the destructor function is called for each object in the collection).

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

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 (for example: **somf_TDictionary_somfDeleteAll**). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you 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.

Return Value

None.

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** or **somf_TDictionary** or any of the other classes that inherit from **somf_TCollection**.

Original Class

somf_TCollection

somfIsEqual Method

Purpose

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

IDL Syntax

boolean somfIsEqual(in somf_MCollectible *obj***);**

Description

The **somfIsEqual** 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, etc.

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** 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.

Original Class

somf_MCollectible (overridden here)

somfMember Method

Purpose

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 (for 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** that may or may not be a member of the collection.

Return Value

Two possible return values are valid for this method:

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

somfRemove Method

Purpose

Removes an object from a collection.

IDL Syntax

somf_MCollectible somfRemove (in somf_MCollectible *obj*);

Description

The **somfRemove** method removes a specified object `obj` 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. **somfRemove** is a method name declared in multiple parents (for example: **somf_TCollection**, **somf_THashTable**, **somf_TIterator**, etc.) You will probably have to fully qualify the method name (for example: **somf_TDictionary_somfRemove**). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you can reference this method as:

```
d->somfRemove(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** to be removed from the collection.

Return Value

Two possible return values are valid for this method:

somf_MCollectible
A pointer to the object that was removed.

SOMF_NIL Indicates the specified object was not found.

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** or **somf_TDictionary** or any of the other classes that inherit from **somf_TCollection**.

Original Class

somf_TCollection

Related Information

Methods: somfRemoveAll

somfRemoveAll Method

Purpose

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 (for example: **somf_TDictionary_somfRemoveAll**). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you 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.

Return Value

None.

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** or **somf_TDictionary** or any of the other classes that inherit from **somf_TCollection**.

Original Class

somf_TCollection

Related Information

Methods: somfRemove

somfSetTestFunction Method

Purpose

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 **somfIsEqual** or **somfIsSame** method.

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

Return Value

None.

Original Class

somf_TCollection

Related Information

Methods: somfTestFunction

somfTCollectionInit Method

Purpose

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 **somfIsEqual** or **somfIsSame** method.

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

Return Value

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

Original Class

somf_TCollection

somfTestFunction Method

Purpose

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 **somfIsSame** or **somfIsEqual**.

Original Class

somf_TCollection

Related Information

Methods: somfSetTestFunction

somf_TDeque Class

Description

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

- A *queue* is a list where the elements are inserted and removed using a first-in, first-out (FIFO) approach.
- A *stack* is a list where the elements are inserted and removed using a last-in, first-out (LIFO) approach.
- A *deque* is a double-ended queue that permits insertion and removal at either end of the list.

All three of these data structures are implemented in the **somf_TDeque** class, with different methods processing the logically different structures. However, the **somf_TDeque** class is more than all three data structures combined, because objects can be inserted and removed from any point in the **somf_TDeque**. In addition, the **somf_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: **somtk**

Objects of class **somf_MCollectible** that are inserted into a **somf_TDeque** collection could override the **somfIsSame** method.

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

somf_TSequence

Metaclass

SOMClass

Ancestor Classes

somf_TSequence, somf_TCollection, somf_MCollectible, SOMObject

New Methods

somfAddAfter
somfAddBefore
somfAddLast

somfAddFirst
somfRemoveLast
somfRemoveFirst
somfCreateSequenceIterator
somfRemoveQ
somfInsert
somfPop
somfPush
somfCreateNewLink
somfAssign
somfTDequeInitF
somfTDequeInitD

Overriding Methods

somInit
somUninit
somfAdd
somfRemove
somfDeleteAll
somfRemoveAll
somfCount
somfAfter
somfBefore
somfLast
somfFirst
somfMember
somfCreateIterator

somfAdd Method

Purpose

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 (overridden here)

Related Information

Methods: somfAddAfter, somfAddBefore, somfAddLast, somfAddFirst

somfAddAfter Method

Purpose

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, *existobj.*

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.

Return Value

None.

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

Related Information

Methods: somfAdd, somfAddBefore, somfAddLast, somfAddFirst

somfAddBefore Method

Purpose

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.

Return Value

None.

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

Related Information

Methods: somfAdd, somfAddAfter, somfAddLast, somfAddFirst

somfAddFirst Method

Purpose

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.

Return Value

None.

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

Related Information

Methods: somfAdd, somfAddAfter, somfAddBefore, somfAddLast

somfAddLast Method

Purpose

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.

Return Value

None.

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

Related Information

Methods: somfAdd, somfAddAfter, somfAddBefore, somfAddFirst

somfAfter Method

Purpose

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 the **somf_MCollectible** that is in front of the returned `obj` .

Return Value

There are two possible valid return values for this method:

somf_MCollectible
A pointer to the 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 (overridden here)

Related Information

Methods: somfBefore, somfFirst, somfLast

somfAssign Method

Purpose

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/resets the instance variables of the receiver to the values of the source. This operation is logically equivalent to using the "=" operator.

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

Return Value

None.

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

somfBefore Method

Purpose

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

There are two possible valid return values for this method:

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

SOMF_NIL The `obj` is the first object in 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 (overridden here)

Related Information

Methods: somfAfter, somfFirst, somfLast

somfCount Method

Purpose

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 (for 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 (overridden here)

somfCreateIterator Method

Purpose

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

IDL Syntax

somf_TIterator somfCreateIterator ( );

Description

The **somfCreateIterator** 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_TDequeIterator** to point to an instance of a **somf_TDeque**. One other way is to use the **somf_TDequeIterator**'s initializer method described on page 97. The final way is to use **somf_TDeque**'s method **somfCreateSequenceIterator,** described on page 72.

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 (overridden here)

Related Information

Methods: somfCreateSequenceIterator

somfCreateNewLink Method

Purpose

Creates a new link in a **somf_TDeque** collection, given two objects as the previous and next **somf_TDequeLinkable** 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

somfCreateSequenceIterator Method

Purpose

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

IDL Syntax

somf_TSequenceIterator somfCreateSequenceIterator ( );

Description

The **somfCreateSequenceIterator** 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_TDequeIterator** to point to an instance of a **somf_TDeque**. One other way is to use **somf_TDequeIterator**'s initializer method described on page 97. The final way is to use **somf_TDeque**'s **somfCreateIterator** method described on page 70.

This method is identical to **somfCreateIterator**; you could use either one. The only difference is that the type of the return value for this method is a **somf_TSequenceIterator**, and the return type for the return value of **somfCreateIterator** is a **somf_TIterator**. However, both methods return an instance of a **somf_TDequeIterator** 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

Related Information

Methods: somfCreateIterator

somfDeleteAll Method

Purpose

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** to remove the objects from the collection and deleting them some other way.

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

Return Value

None.

Example

```
somf_TDeque dq;
Environment *ev;

ev = somGetGlobalEnvironment();

dq = somf_TDequeNew();

/* Add some objects to dq */

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

_somFree (dq);
```

Original Class

somf_TCollection (overridden here)

somfFirst Method

Purpose

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 (for example: **somf_TSequence**, **somf_TIterator**, etc.). You will probably have to fully qualify the method name (for example: **somf_TDeque_somfFirst**). This is the only way the linker can tell them apart.

This is not a problem in C++. In C++ you 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

There are two possible valid return values for this method:

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 (overridden here)

Related Information

Methods: somfAfter, somfBefore, somfLast

somfInsert Method

Purpose

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** 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.

Return Value

None.

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

Related Information

Methods: somfRemoveQ

somfLast Method

Purpose

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_TSequenceIterator** 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

There are two possible valid return values for this method:

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

SOMF_NIL Nothing is in the collection.

Example

```
somf_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 (overridden here)

Related Information

Methods: somfAfter, somfBefore, somfFirst

somfMember Method

Purpose

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 (for 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

There are two possible valid return values for this method:

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

SOMF_NIL
: `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 (overridden here)

somfPop Method

Purpose

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/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

Related Information

Methods: somfPush

somfPush Method

Purpose

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/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.

Return Value

None.

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

Related Information

Methods: somfPop

somfRemove Method

Purpose

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 (for example: **somf_TCollection**, **somf_THashTable**, **somf_TIterator**, etc.) You will probably have to fully qualify the method name (for example: **somf_TDeque_somfRemove**). 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

There are two possible valid return values for this method:

somf_MCollectible
: A pointer to the object that was removed.

SOMF_NIL
: The object was not found.

Example

```
somf_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 (overridden here)

Related Information

Methods: somfRemoveFirst, somfRemoveLast, somfRemoveAll

somfRemoveAll Method

Purpose

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 (for example: **somf_TDeque_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_TDeque.**
ev	A pointer to the **Environment** structure for the calling method.

Return Value

None.

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 (overridden here)

Related Information

Methods: somfRemove, somfRemoveFirst, somfRemoveLast

somfRemoveFirst Method

Purpose

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

There are two possible valid return values for this method:

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

Related Information

Methods: somfRemoveLast, somfRemove, somfRemoveAll

somfRemoveLast Method

Purpose

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

There are two possible valid return values for this method:

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

Related Information

Methods: somfRemoveFirst, somfRemove, somfRemoveAll

somfRemoveQ Method

Purpose

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

There are two possible valid return values for this method:

somf_MCollectible
A pointer to the object that was removed.

SOMF_NIL The object was not found.

Example

```
somf_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

Related Information

Methods: somfInsert

somfTDequeInitD Method

Purpose

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

Related Information

Methods: somfTDequeInitF

somfTDequeInitF Method

Purpose

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 **somfIsEqual** or **somfIsSame** method.

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

_somFree (dq1);
```

Original Class

somf_TDeque

Related Information

Methods: somfTDequeInitD

somf_TDequeIterator Class

Description

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_TSequenceIterator

Metaclass

SOMClass

Ancestor Classes

somf_TSequenceIterator, somf_TIterator, SOMObject

New Methods

somfTDequeIteratorInit

Overriding Methods

somfFirst
somfNext
somfLast
somfPrevious
somfRemove

somfFirst Method

Purpose

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. 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, this also revalidates the iterator.

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

```
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

This method returns a pointer to the first **somf_MCollectible** object in the collection. Or, 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 (overridden here)

Related Information

Methods: somfNext

somfLast Method

Purpose

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**, etc.). You will probably have to fully qualify the method name (for example: **somf_TDequeIterator_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:

```
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 (overridden here)

Related Information

Methods: somfPrevious

somfNext Method

Purpose

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 (for example: **somf_TDequeIterator_somfNext**). 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** collection has changed (other than through the use of the **somfRemove** method of this iterator) since the last time the **somfFirst** or **somfLast** method was called, the iterator becomes invalid and will *fail* when 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 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

There are two possible valid return values for this method:

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

SOMF_NIL
: The end of the collection has been reached.

Example

```
somf_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 (overridden here)

Related Information

Methods: somfFirst

somfPrevious Method

Purpose

Gets the previous object in a deque collection.

IDL Syntax

somf_MCollectible somfPrevious ( );

Description

The **somfPrevious** method determines the previous object in the deque collection that corresponds to the deque iterator represented by the receiving object. The method also 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. Thus, if any child of **somf_TSequenceIterator** is used with **somf_TPrimitiveLinkedListIterator**, then the name of the method must be fully qualified (for example: **somf_TDequeIterator_somfPrevious**). 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->somfPrevious(ev);
```

If the **somf_TDeque** collection changes while using this iterator, the iterator becomes invalid and will *fail* if asked to find the previous object. For example, if the collection's **somfAdd** method is called after starting to iterate through the collection, the iterator will not allow iteration to continue. The iterator must be reset, and the easiest way to do that is to call the iterator's **somfLast** method and start over.

Parameters

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

Return Value

There are two possible valid return values for this method:

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

SOMF_NIL The beginning of the collection has been reached.

Example

```
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 itrobj to point to the next to the last object in dq */
somf_TDequeIterator_somfLast(itr,ev);
itrobj = _somfPrevious(itr,ev);

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

Original Class

somf_TSequenceIterator (overridden here)

Related Information

Methods: somfLast

somfRemove Method

Purpose

Removes the current object from a deque collection.

IDL Syntax

void somfRemove ( );

Description

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

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

C cannot handle methods from different classes having the same name when they inherit the name from different parents. **somfRemove** is a method name declared in multiple parents (for example: **somf_TCollection**, **somf_THashTable**, **somf_TIterator**, etc.) You will probably have to fully qualify the method name (for example: **somf_TDequeIterator_somfRemove**). 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->somfRemove(ev);
```

If the **somf_TDeque** collection has changed (other than through the use of the **somfRemove** method of this iterator) since the last time the **somfFirst** or **somfLast** method was called, the iterator becomes invalid and will *fail* if asked to remove an 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** or **somfLast** 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

None.

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 (overridden here)

somfTDequeIteratorInit Method

Purpose

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

IDL Syntax

somf_TDequeIterator somfTDequeIteratorInit (in somf_TDeque *h***);**

Description

The **somfTDequeIteratorInit** method initializes an iterator of class **somf_TDequeIterator**, given the **somf_TDeque** collection over which iteration is needed.

Note: This is one of three ways to initialize a **somf_TDequeIterator** to point to an instance of a **somf_TDeque** collection. One other way is to use the **somfCreateIterator** method of the **somf_TDeque** class, as described on page 70. The final way is to use the **somf_TDeque** class's **somfCreateSequenceIterator** method, described on page 72.

Note: You cannot override this method

Parameters

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

somf_TDequeLinkable Class

Description

The **somf_TDequeLinkable** class is a subclass of **somf_MLinkable**. It provides a generic **somf_MLinkable** class to contain **somf_MCollectible**. An object of class **somf_TDequeLinkable** is used (transparently) by the **somf_TDeque** class for each node of a deque collection. The **somf_TDequeLinkable** object provides the “linkability” (that is, the left and right links) to its two adjacent nodes in the collection.

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

somf_MLinkable

Metaclass

SOMClass

Ancestor Classes

somf_MLinkable, SOMObject

New Methods

somfGetValue
somfSetValue
somfTDequeLinkableInitDDM
somfTDequeLinkableInitDD

Overriding Methods

somInit

somfGetValue Method

Purpose

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** 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

Methods: somfSetValue

somfSetValue Method

Purpose

Sets the value 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.

Return Value

None.

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

Methods: somfGetValue

somfTDequeLinkableInitDD Method

Purpose

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 dl1;
Environment *ev;

ev = somGetGlobalEnvironment();

dl1 = somf_TDequeLinkableNew();
_somfTDequeLinkableInitDD(dl1, ev, SOMF_NIL, SOMF_NIL);

_somFree (dl1);
```

Original Class

somf_TDequeLinkable

Related Information

Methods: somfTDequeLinkableInitDDM

somfTDequeLinkableInitDDM Method

Purpose

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** node before this one.
next	A pointer to the **somf_TDequeLinkable** node after 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

Methods: somfTDequeLinkableInitDD