OS/390

SOMobjects Programmer's Reference, Volume 3

SC28-1999-01

OS/390

SOMobjects Programmer's Reference, Volume 3

SC28-1999-01

Note

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

Second Edition, June 1997

This is a major revision of, and obsoletes, SC28-1999-00.

This edition applies to Version 2 Release 4 of OS/390 (5647-A01) and to all subsequent releases and modifications until otherwise indicated in new editions.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address below.

IBM welcomes your comments. A form for readers' comments may be provided at the back of this publication, or you may address your comments to the following address:

International Business Machines Corporation
Department 55JA, Mail Station P384
522 South Road
Poughkeepsie, NY 12601-5400
United States of America

FAX (United States & Canada): 1+914+432-9405
FAX (Other Countries):
Your International Access Code +1+914+432-9405

IBMLink (United States customers only): KGNVMC(MHVRCFS)
IBM Mail Exchange: USIB6TC9 at IBMMAIL
Internet: mhvrcfs@vnet.ibm.com

If you would like a reply, be sure to include your name, address, telephone number, or FAX number.

Make sure to include the following in your comment or note:

- Title and order number of this book
- Page number or topic related to your comment

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Notices

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

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

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 inquires, in writing, to:

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

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

IBM Corporation
Mail Station P300
522 South Road
Poughkeepsie, NY 12601-5400
USA
Attention: Information Request

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

Programming Interface Information

This publication is intended to help the customer use SOMobjects to build object-oriented class libraries. This publication documents General-use Programming Interface and Associated Guidance Information provided by SOMobjects.

General-use programming interfaces allow the customer to write programs that obtain the services of SOMobjects.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States and/or other countries:

AFP
AnyNet
BookManager
C/370

C++/MVS
CICS/ESA
CICS/OS2
CICSPlex
C/MVS
DB2
DFSMSdfp
DFSMSdss
DFSMShsm
DFSMS/MVS
DFSMSrmm
DFSORT
ESCON
Extended Services
FFST
FFST/MVS
GDDM
Hardware Configuration Definition
Hiperbatch
IBM
IBMLink
IMS
IMS/ESA
Language Environment
MVS/ESA
NetView
OPC
Open Class
OpenEdition
OS/2
OS/390
Parallel Sysplex
PSF
RACF
RMF
S/390
SOMobjects
SP
System/390
SystemView
VisualLift
VTAM

The following terms, **DENOTED BY A DOUBLE ASTERISK (**)**, used in this publication, are trademarks of other companies as follows:

UNIX	X/Open Company Limited
Windows	Microsoft Corporations

About This Book

This book provides reference material for SOMobjects. It contains a reference page for every class and method that the SOMobjects runtime library provides. It contains syntax and a brief description for every method that the SOMobjects runtime library provides.

Who Should Use This Book

This book is for programmers who:

- Use SOMobjects to build object-oriented class libraries.
- Code application programs that use SOMobjects class libraries or SOMobjects frameworks.

This book assumes that the reader is an experienced programmer who is familiar with basic concepts of object-oriented programming. Practical experience with an object-oriented programming language is helpful but not essential.

How This Book Is Organized

This book describes the classes and methods that are part of the Collection Classes.

The description of a SOMobjects class contains the following information:

Purpose	The purpose of the class.
Member Name	The member name associated with the IDL interface (.IDL) file and usage binding (.h/.xh/.hh) files for the class. The member resides in the partitioned data set (PDS) that contains the SOMobjects IDL or header library.
Import Name	The import name associated with the definition side deck needed by the prelink and link job for the class.
Base Class	The class's direct base (parent) classes.
Metaclass	The class's metaclass.
Ancestor Classes	The class's ancestor (indirect base) classes.
Subclasses	Classes that inherit attributes of parent classes; also called *child classes.*
Types	Actual data types; descriptions of what each class looks like to the user and the compiler.
Attributes	The content of a class or object. Also called *data*.
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 description of a SOMobjects method contains the following information:

Purpose	The purpose of the method.

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.
Parameters	Descriptions of parameters for each method.
Restrictions and Limitations	Descriptions of restrictions and limitations associated with calling each method.
Returned Values	A description of the method's return value.
Example	An example of using or overriding the method, written in C or C++ language. (Methods of SOMobjects classes can be issued from any language that can use SOMobjects.) See *OS/390 SOMobjects Programmer's Guide* for information about SOMobjects naming conventions.
Original Class	The name of the class that introduces the method (the class is documented separately in this book).
Related Information	Related methods and functions (and macros, for the SOMobjects kernel) that can be found in this book.

Where to Find More Information

The book references the following publications using the shortened version of the book title. The following table lists the shortened titles, complete titles, and order numbers of the books you might need while you are using this book.

Table 1. SOMobjects Publications

Short Title Used in This Book	Title	Order Number
OS/390 SOMobjects: Getting Started	*OS/390 SOMobjects: Getting Started*	GA22-7248
OS/390 SOMobjects Messages, Codes, and Diagnosis	*OS/390 SOMobjects Messages, Codes, and Diagnosis*	SC28-1996
OS/390 SOMobjects Object Services	*OS/390 SOMobjects Object Services*	SC28-1995
OS/390 SOMobjects Programmer's Reference, Volume 1	*OS/390 SOMobjects Programmer's Reference, Volume 1*	SC28-1997
OS/390 SOMobjects Programmer's Reference, Volume 2	*OS/390 SOMobjects Programmer's Reference, Volume 2*	SC28-1998
OS/390 SOMobjects Configuration and Administration Guide	*OS/390 SOMobjects Configuration and Administration Guide*	GC28-1851
OS/390 SOMobjects Programmer's Guide	*OS/390 SOMobjects Programmer's Guide*	GC28-1859

CORBA Publications

- *The Common Object Request Broker: Architecture and Specification*, published by the Object Management Group and X/Open.

C++ Publications

- *Algorithms in C++* published by Addison-Wesley.

Summary of Changes

Summary of Changes
for SC28-1999-01
OS/390 Release 4

This book contains information previously presented in *OS/390 SOMobjects Programmer's Reference, Volume 3*, SC28-1999-00, which supports Version 1 Release 3.

This book includes terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

New Information

- Import Name was added to each class.
- Part 1. Inheritance Hierarchy of the Collection Classes

Inheritance Hierarchy of the Collection Classes

The inheritance hierarchy for the collection classes is depicted in the following figure. The diagram does not illustrate all of the classes, only those that have some position in the inheritance hierarchy of the set. Classes not shown include somf_TPrimitiveLinkedList, somf_TPrimitiveLinkedListIterator, and somf_TsortedSequenceNode; all of which inherit directly from SOMObject.

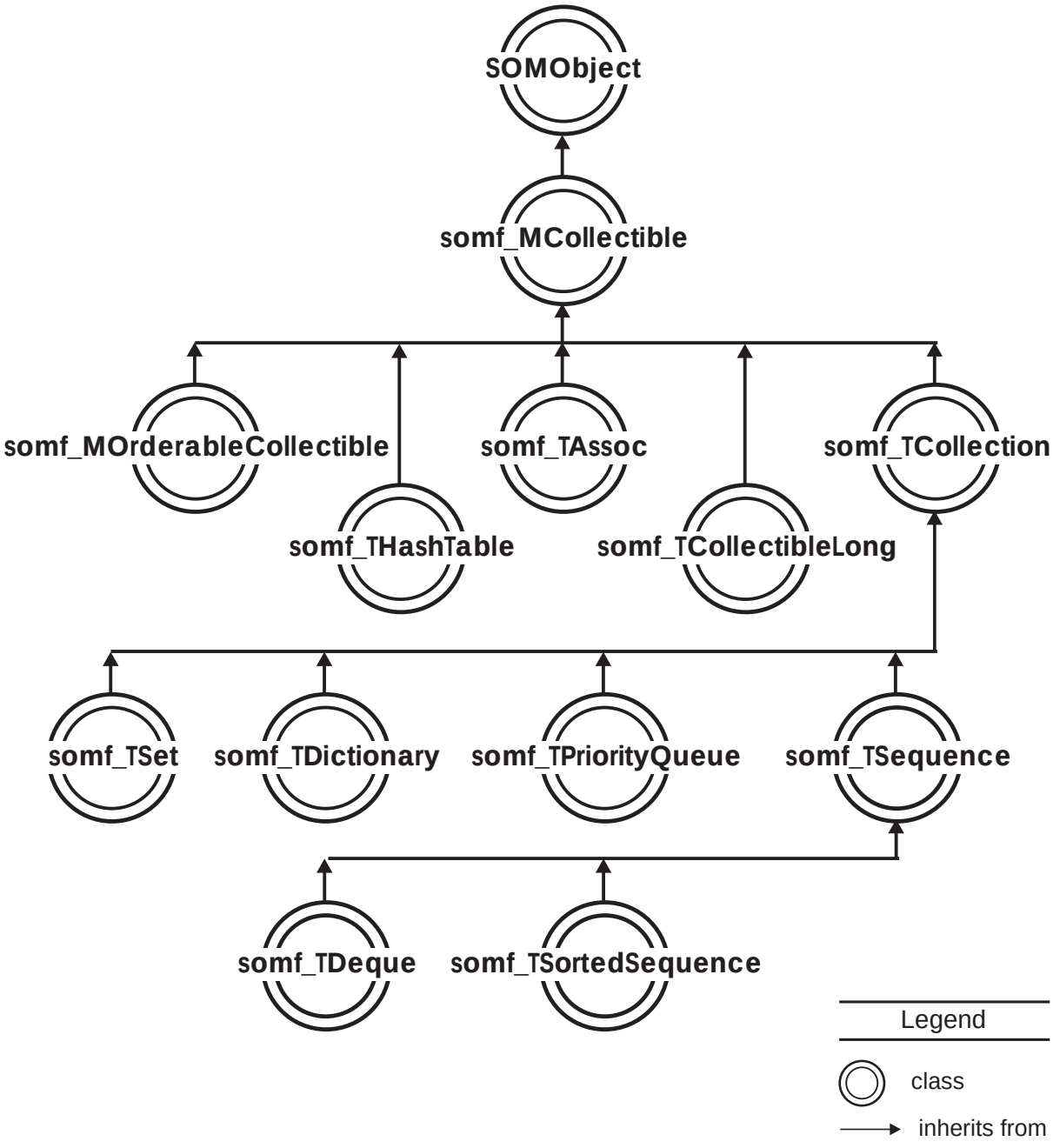

Figure 1. Inheritance Hierarchy of Collection Classes, Part One

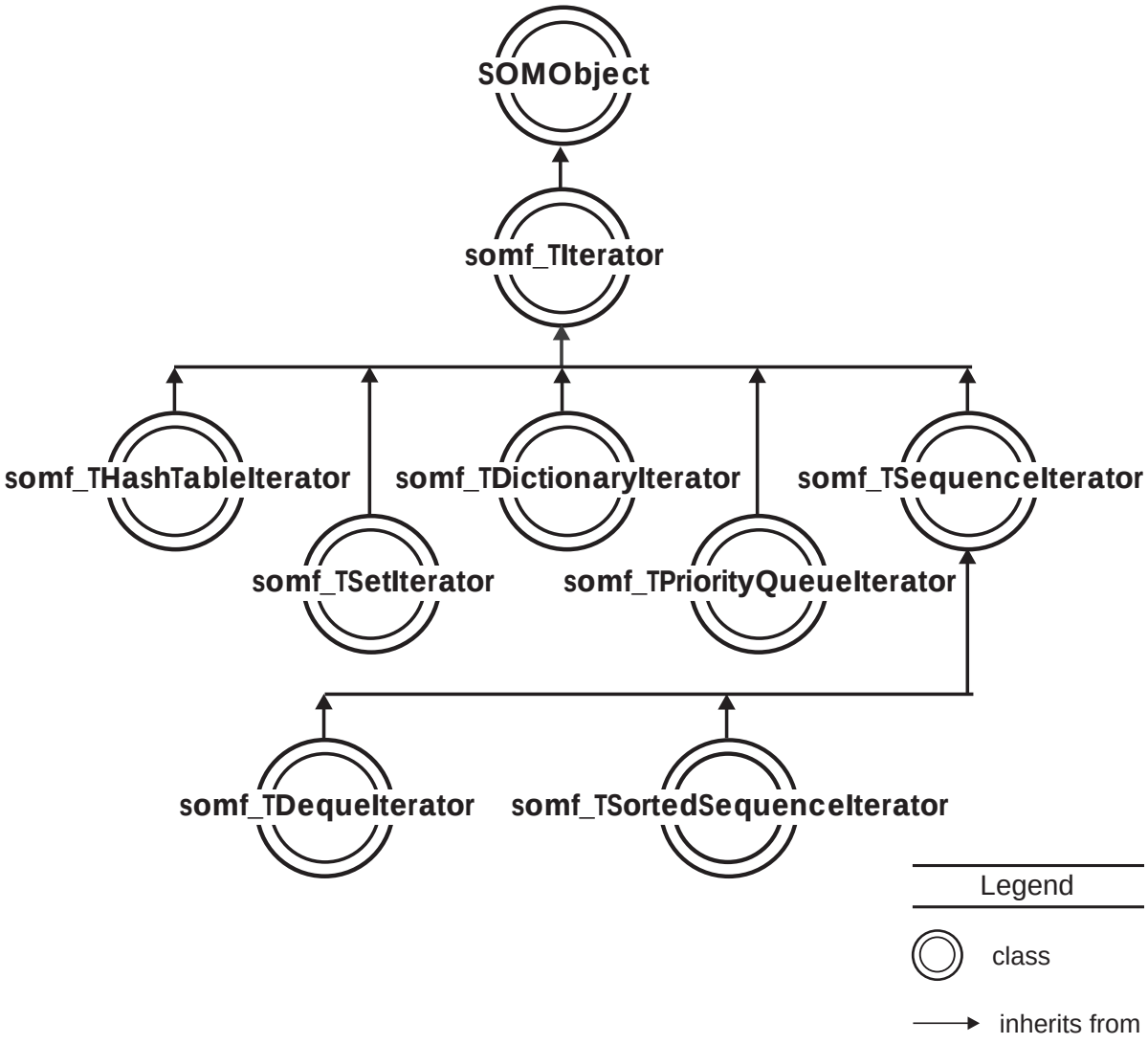

Figure 2. Inheritance Hierarchy of Collection Classes, Part Two

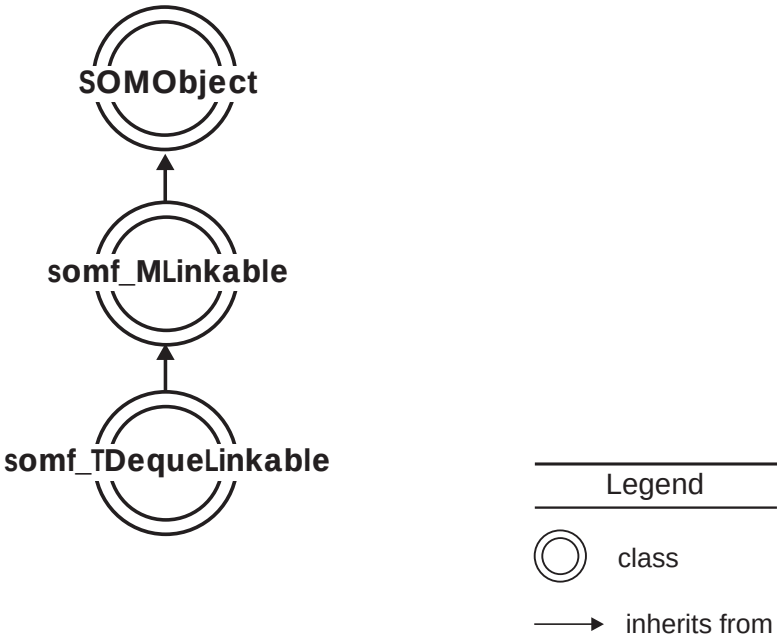

Figure 3. Inheritance Hierarchy of Collection Classes, Part Three

Chapter 1. somf_MCollectible Class

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

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

Member Name:

mcollect

Import Name:

GOSSOMUC

Base Class:

SOMObject

Metaclass:

SOMClass

Ancestor Classes:

SOMObject

Subclasses:

None.

Types:

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.

Attributes:

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.

New Methods:

somfClone
somfClonePointer
somfHash
somfIsEqual
somfIsSame
somfIsNotEqual

Overriding Methods:

None.

somfClone Method

Purpose

Provides a general polymorphic duplication operation.

Syntax

```
somf_MCollectible  somfClone ();
```

Parameters

receiver A pointer to an object of class somf_MCollectible.

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

Restrictions and Limitations

None.

Returned Values

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

somfClonePointer

somfClonePointer Method

Purpose

Returns a pointer to a Clone.

Note: You cannot override this method.

Syntax

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

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.

Restrictions and Limitations

None.

Returned Values

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

somfClone

somfHash Method

Purpose

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

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

Syntax

```
long somfHash ();
```

Parameters

receiver A pointer to an object of class somf_MCollectible.

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

Restrictions and Limitations

None.

Returned Values

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

Related Information

None.

somfIsEqual Method

Purpose

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

Most utility classes allow you to specify what methods to use when comparing objects for insertion, deletion. The choice is to use either the somfIsEqual method, or 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.

Syntax

```
boolean  somfIsEqual(
    in somf_MCollectible 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.

Restrictions and Limitations

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.

Returned Values

This method returns a boolean value:

TRUE *Obj* is equal to the receiving object.

FALSE *Obj* is not equal to the receiving object.

Example

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

somfIsNotEqual

somfIsNotEqual Method

Purpose

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

Syntax

```
boolean  somfIsNotEqual(
    in somf_MCollectible 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.

Restrictions and Limitations

None.

Returned Values

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

somfIsEqual

somfIsSame Method

Purpose

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

Syntax

```
boolean  somfIsSame(
    in somf_MCollectible 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.

Restrictions and Limitations

None.

Returned Values

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

Related Information

None.

Chapter 2. somf_MLinkable Class

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

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.

Member Name:

mlink

Import Name:

GOSSOMUC

Base Class:

SOMObject

Metaclass:

SOMClass

Ancestor Classes:

SOMObject

Subclasses:

None.

Types:

None.

Attributes:

None.

New Methods:

somfGetNext
somfGetPrevious
somfMLinkableInit
somfSetNext
somfSetPrevious

Overriding Methods:

somDefaultInit

somfGetNext Method

Purpose

Gets a pointer to the next somf_MLinkable object.

Syntax

```
somf_MLinkable  somfGetNext ();
```

Parameters

receiver A pointer to an object of class somf_MLinkable.

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

Restrictions and Limitations

None.

Returned Values

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

somfSetNext
somfSetPrevious

somfGetPrevious Method

Purpose

Gets a pointer to the previous somf_MLinkable object.

Syntax

```
somf_MLinkable  somfGetPrevious ();
```

Parameters

receiver A pointer to an object of class somf_MLinkable.

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

Restrictions and Limitations

None.

Returned Values

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

somfGetNext
somfSetPrevious

somfMLinkableInit Method

Purpose

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

Note: You cannot override this method.

Syntax

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

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.

Restrictions and Limitations

None.

Returned Values

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

Related Information

None.

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.

Syntax

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

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.

Restrictions and Limitations

None.

Returned Values

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

somfGetNext
somfSetPrevious

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.

Syntax

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

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.

Restrictions and Limitations

None.

Returned Values

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

somfGetPrevious
somfSetNext

Chapter 3. somf_MOrderableCollectible Class

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

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

Member Name:

morder

Import Name:

GOSSOMUC

Base Class:

somf_MCollectible

Metaclass:

SOMClass

Ancestor Classes:

somf_MCollectible
SOMObject

Subclasses:

None.

Types:

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.

Attributes:

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.

New Methods:

somfIsGreaterThan
somfIsLessThan
somfCompare
somfIsGreaterThanOrEqualTo
somfIsGreaterThanOrEqualTo

somfCompare Method

Purpose

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

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

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

Syntax

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

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.

Restrictions and Limitations

None.

Returned Values

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) == \n
somf_MOrderableCollectible_kLessThan)
     somPrintf(" a1 is less than a2);
else
     somPrintf(" a1 is NOT less than a2);
if ( _somfCompare(a1,ev,a2) == \n
somf_MOrderableCollectible_kGreaterThan)
     somPrintf(" a2 is greater than a1);
else
     somPrintf(" a2 is NOT greater than a1");
if ( _somfCompare(a2,ev,a2) == somf_MOrderableCollectible_kEqual)
     somPrintf(" a2 is equal a2);
else
     somPrintf(" a2 is NOT equal to a2");
_somFree (a1);
_somFree (a2);
```

Original Class

somf_MOrderableCollectible

Related Information

somfIsEqual
somfIsGreaterThan
somfIsLessThan

somfIsGreaterThan Method

Purpose

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

This method must be overridden if a class inherits from somf_MOrderableCollectible. If not, an error message is written and processing will end.

Syntax

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

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.

Restrictions and Limitations

You cannot use this method directly from this class; it must be overridden.

Returned Values

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

Example

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

somfIsEqual
somfIsGreaterThanOrEqualTo
somfIsLessThan

somfIsGreaterThanOrEqualTo Method

Purpose

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

Syntax

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

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.

Restrictions and Limitations

None.

Returned Values

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

somfIsEqual
somfIsGreaterThan
somfIsLessThan

somfIsLessThan Method

Purpose

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

This method must be overridden if a class inherits from somf_MOrderableCollectible. If not, an error message is written and processing will end.

Syntax

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

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.

Restrictions and Limitations

You cannot use this method directly from this class; it must be overridden.

Returned Values

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

Example

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

somfIsEqual
somfIsGreaterThan
somfIsLessThanOrEqualTo

somfIsLessThanOrEqualTo Method

Purpose

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

Syntax

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

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.

Restrictions and Limitations

None.

Returned Values

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

somfIsLessThan
somfIsGreaterThan
somfIsEqual

Chapter 4. somf_TAssoc Class

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 Class and somf_TDictionary Class use 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.

This class is not thread-safe.

Member Name:

tassoc

Import Name:

GOSSOMUC

Base Class:

somf_MCollectible

Metaclass:

SOMClass

Ancestor Classes:

somf_MCollectible
SOMObject

Subclasses:

None.

Types:

None.

Attributes:

None.

New Methods:

somfGetKey
somfGetValue
somfSetKey
somfSetValue
somfTAssocInitM
somfTAssocInitMM

Overriding Methods:

somDefaultInit
somDestruct

somfGetKey Method

Purpose

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

Note: You cannot override this method.

Syntax

somf_MCollectible **somfGetKey** ();

Parameters

receiver A pointer to an object of class somf_TAssoc.

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

Restrictions and Limitations

None.

Returned Values

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

somfSetKey
somfSetValue
somfGetValue

somfGetValue Method

Purpose

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

Note: You cannot override this method.

Syntax

```
somf_MCollectible somfGetValue ();
```

Parameters

receiver A pointer to an object of class somf_TAssoc.

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

Restrictions and Limitations

None.

Returned Values

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

somfSetValue
somfSetKey
somfGetKey

somfSetKey Method

Purpose

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

Note: You cannot override this method.

Syntax

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

Parameters

receiver A pointer to an object of class somf_TAssoc.

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

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

Restrictions and Limitations

None.

Returned Values

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

somfGetKey
somfSetValue
somfGetValue

somfSetValue Method

Purpose

Sets the value to an associated (key, value) pair represented by the receiving object.

Note: You cannot override this method.

Syntax

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

Parameters

receiver A pointer to an object of class somf_TAssoc.

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

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

Restrictions and Limitations

None.

Returned Values

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

somfGetValue
somfGetKey
somfSetKey

somfTAssocInitM Method

Purpose

Initializes a somf_TAssoc object to a given key (k). The value (v) is set to SOMF_NIL. An object of class somf_TAssoc is a pair.

Note: You cannot override this method.

Syntax

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

Parameters

receiver A pointer to an object of class somf_TAssoc.

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

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

Restrictions and Limitations

None.

Returned Values

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

somfTAssocInitMM

somfTAssocInitMM Method

Purpose

Initializes a somf_TAssoc object to a given key (k) and value (v). An object of class somf_TAssoc is a pair.

Note: You cannot override this method.

Syntax

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

Parameters

receiver A pointer to an object of class somf_TAssoc.

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

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

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

Restrictions and Limitations

None.

Returned Values

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

somfTAssocInitM

Chapter 5. somf_TCollectibleLong Class

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

This class is not thread-safe.

This class is reentrant.

Member Name:

tclong

Import Name:

GOSSOMUC

Base Class:

somf_MCollectible

Metaclass:

SOMClass

Ancestor Classes:

somf_MCollectible
SOMObject

Subclasses:

None.

Types:

None.

Attributes:

None.

New Methods:

somfGetValue
somfSetValue
somfTCollectibleLongInit

Overriding Methods:

somDefaultInit
somfIsEqual
somfHash

somfGetValue Method

Purpose

Gets the value of the long in the receiving object.

Syntax

```
long  somfGetValue ();
```

Parameters

receiver A pointer to an object of class somf_TCollectibleLong.

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

Restrictions and Limitations

None.

Returned Values

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

somfSetValue
somfIsEqual

somfHash Method

Purpose

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

Syntax

```
long  somfHash ();
```

Parameters

receiver A pointer to an object of class somf_TCollectibleLong.

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

Restrictions and Limitations

None.

Returned Values

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

Related Information

somfGetValue
somfSetValue
somfTCollectibleLongInit
somfIsEqual

somfIsEqual Method

Purpose

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

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 and so forth.

Syntax

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

Parameters

receiver A pointer to an object of class somf_TCollectibleLong.

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

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

Restrictions and Limitations

None.

Returned Values

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

Related Information

somfGetValue
somfSetValue
somfTCollectibleLongInit

somfSetValue Method

Purpose

Sets the long value in an object of class somf_TCollectibleLong.

Syntax

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

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.

Restrictions and Limitations

None.

Returned Values

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

somfGetValue
somfIsEqual
somfTCollectibleLongInit

somfTCollectibleLongInit Method

Purpose

Initializes a new somf_TCollectibleLong object.

Note: You cannot override this method.

Syntax

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

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.

Restrictions and Limitations

None.

Returned Values

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

somfSetValue
somfGetValue
somfIsEqual
somfHash

Chapter 6. somf_TCollection Class

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

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

Note: The somf_TCollection class uses the 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.

Member Name:

tcollect

Import Name:

GOSSOMUC

Base Class:

somf_MCollectible

Metaclass:

SOMClass

Ancestor Classes:

somf_MCollectible
SOMObject

Subclasses:

None.

Types:

None.

Attributes:

None.

New Methods:

somfAdd
somfAddAll
somfRemove
somfRemoveAll
somfDeleteAll

somfCount
somfMember
somfCreateIterator
somfTestFunction
somfSetTestFunction
somfTCollectionInit

Overriding Methods:

somfIsEqual

somfAdd Method

Purpose

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.

Syntax

```
somf_MCollectible  somfAdd(
    in somf_MCollectible 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 an object of class somf_MCollectible that will be added to the receiving object.

Restrictions and Limitations

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.

Returned Values

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

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

Example

For examples of how this method looks when it is invoked, see somf_TDeque Class, somf_TDictionary Class, or any of the other classes that inherit from somf_TCollection.

Original Class

somf_TCollection

Related Information

somfAddAll

somfAddAll Method

Purpose

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.

Syntax

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

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.

Restrictions and Limitations

None.

Returned Values

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

somfAdd

somfCount Method

Purpose

Gets the number of objects in the collection represented by the receiving object, and returns that number.

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

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

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

Syntax

```
long  somfCount ();
```

Parameters

receiver A pointer to an object of class somf_TCollection.

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

Restrictions and Limitations

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.

Returned Values

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

Example

For examples of how this method looks when it is invoked, see somf_TDeque Class or somf_TDictionary Class or any of the other classes that inherit from somf_TCollection.

Original Class

somf_TCollection

Related Information

None.

somfCreateIterator Method

Purpose

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

Every class that inherits from the somf_TCollection class mustoverride this method for that class to work correctly.

Syntax

```
somf_TIterator  somfCreateIterator ();
```

Parameters

receiver A pointer to an object of class somf_TCollection.

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

Restrictions and Limitations

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.

Returned Values

This method returns a pointer to the new iterator.

Example

For examples of how this method looks when it is invoked, see somf_TDeque Class or somf_TDictionary Class or any of the other classes that inherit from somf_TCollection.

Original Class

somf_TCollection

Related Information

None.

somfDeleteAll Method

Purpose

Removes all of the objects from the receiving object and deallocates the storage that these objects might have owned.

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

Every class that inherits from the somf_TCollection class mustoverride this method for that class to work correctly.

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

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

Syntax

void **somfDeleteAll** ();

Parameters

receiver A pointer to an object of class somf_TCollection.

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

Restrictions and Limitations

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.

Returned Values

None.

Example

For examples of how this method looks when it is invoked, see somf_TDeque Class or somf_TDictionary Class or any of the other classes that inherit from somf_TCollection.

Original Class

somf_TCollection

Related Information

None.

somfIsEqual Method

Purpose

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

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

Syntax

```
boolean somfIsEqual(
    in somf_MCollectible 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 somf_MCollectible Class object that the receiving object will be compared against.

Restrictions and Limitations

None.

Returned Values

TRUE *Obj* is equal to the receiving object.

FALSE *Obj* is not equal to the receiving object.

Example

None.

Original Class

somf_MCollectible

Related Information

None.

somfMember Method

Purpose

Gets an *obj* in the collection.

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

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

Syntax

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

Parameters

receiver A pointer to an object of class somf_TCollection.

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

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

Restrictions and Limitations

None.

Returned Values

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

Related Information

None.

somfRemove Method

Purpose

Removes an object from a collection.

Every class that inherits from the somf_TCollection class mustoverride 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. You will probably have to fully qualify the method name. This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you can reference this method as:

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

Syntax

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

Parameters

receiver A pointer to an object of class somf_TCollection.

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

obj A pointer to the object of class somf_MCollectible Class to be removed from the collection.

Restrictions and Limitations

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.

Returned Values

somf_MCollectible
A pointer to the object that was removed.

SOMF_NIL Indicates the specified object was not found.

Example

For examples of how this method looks when it is invoked, see somf_TDeque Class or somf_TDictionary Class or any of the other classes that inherit from somf_TCollection.

Original Class

somf_TCollection

Related Information

somfRemoveAll

somfRemoveAll Method

Purpose

Removes all of the objects from a collection.

Every class that inherits from the somf_TCollection class mustoverride this method for that class to work correctly.

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

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

Syntax

void **somfRemoveAll** ();

Parameters

receiver A pointer to an object of class somf_TCollection.

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

Restrictions and Limitations

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.

Returned Values

None.

Example

For examples of how this method looks when it is invoked, see somf_TDeque Class or somf_TDictionary Class or any of the other classes that inherit from somf_TCollection.

Original Class

somf_TCollection

Related Information

somfRemove

somfSetTestFunction Method

Purpose

Sets the test method for a collection.

Syntax

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

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 Method or a somfIsSame Method.

This argument should always be set to either

`somf_MCollectibleClassData.somfIsSame or somf_MCollectibleClassData.somfIsEqual`

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

Restrictions and Limitations

None.

Returned Values

None.

Example

None.

Original Class

somf_TCollection

Related Information

somfTestFunction

somfTCollectionInit Method

Purpose

Initializes a new object of class somf_TCollection.

Note: You cannot override this method.

Syntax

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

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 Method or a somfIsSame Method.

This argument should always be set to either

`somf_MCollectibleClassData.somfIsSame or somf_MCollectibleClassData.somfIsEqual`

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

Restrictions and Limitations

None.

Returned Values

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

Example

None.

Original Class

somf_TCollection

Related Information

None.

somfTestFunction Method

Purpose

Determines the method that a collection uses for comparison testing.

Comparison testing is performed on objects already contained in the collection and/or on objects being tested for eligibility.

Syntax

```
somf_MCollectibleCompareFn
somfTestFunction ();
```

Parameters

receiver A pointer to an object of class somf_TCollection.

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

Restrictions and Limitations

None.

Returned Values

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 Method or somfIsEqual Method.

Example

None.

Original Class

somf_TCollection

Related Information

somfSetTestFunction

Chapter 7. somf_TDeque Class

A somf_TDeque class is a child of somf_TSequence Class. 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.

Each 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.

Objects of the somf_MCollectible Class 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 do not 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.

Member Name:

tdeq

Import Name:

GOSSOMUC

Base Class:

somf_TSequence

Metaclass:

SOMClass

Ancestor Classes:

somf_TSequence
somf_TCollection
somf_MCollectible
SOMObject

Subclasses:

None.

Types:

None.

Attributes:

None.

New Methods:

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

Overriding Methods:

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

somfAdd Method

Purpose

Adds an object to a deque collection.

Syntax

```
somf_MCollectible  somfAdd(
    in somf_MCollectible 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 a somf_MCollectible object that will be added to the receiving object.

Restrictions and Limitations

None.

Returned Values

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

Related Information

somfAddAfter
somfAddBefore
somfAddFirst
somfAddLast

somfAddAfter Method

Purpose

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

Syntax

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

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.

Restrictions and Limitations

None.

Returned Values

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

somfAdd
somfAddBefore
somfAddFirst
somfAddLast

somfAddBefore Method

Purpose

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

Syntax

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

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.

Restrictions and Limitations

None.

Returned Values

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

somfAdd
somfAddAfter
somfAddFirst
somfAddLast

somfAddFirst Method

Purpose

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

Syntax

```
void  somfAddFirst(
    in somf_MCollectible 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 object of class somf_MCollectible to be added to the deque.

Restrictions and Limitations

None.

Returned Values

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

somfAdd
somfAddAfter
somfAddBefore
somfAddLast

somfAddLast Method

Purpose

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

Syntax

```
void  somfAddLast(
    in somf_MCollectible 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 object of class somf_MCollectible to be added to the deque.

Restrictions and Limitations

None.

Returned Values

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

somfAdd
somfAddAfter
somfAddBefore
somfAddFirst

somfAfter Method

Purpose

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

Syntax

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

Restrictions and Limitations

None.

Returned Values

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

Related Information

somfBefore
somfFirst
somfLast

somfAssign Method

Purpose

Assigns a deque collection as being equal to a given source deque. That is, the method sets or resets the instance variables of the receiver to the values of the source. This operation is logically equivalent to using the equal (=) operator.

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

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

Syntax

void **somfAssign** (in somf_TDeques);

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.

Restrictions and Limitations

None.

Returned Values

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

Related Information

None.

somfBefore Method

Purpose

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

Syntax

```
somf_MCollectible  somfBefore(
    in somf_MCollectible 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 that is behind the returned *obj*.

Restrictions and Limitations

None.

Returned Values

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

somfAfter
somfFirst
somfLast

somfCount Method

Purpose

Gets the number of objects in a deque collection.

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

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

Syntax

long **somfCount** ();

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.

Restrictions and Limitations

None.

Returned Values

None.

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)

Related Information

None.

somfCreateIterator Method

Purpose

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

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

Syntax

somf_TIterator **somfCreateIterator** ();

Parameters

receiver A pointer to an object of class somf_TDeque.

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

Restrictions and Limitations

None.

Returned Values

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

Related Information

somfCreateSequenceIterator

somfCreateNewLink Method

Purpose

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

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.

Syntax

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

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.

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to the new somf_TDequeLinkable object.

Example

None.

Original Class

somf_TDeque

Related Information

None.

somfCreateSequenceIterator Method

Purpose

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

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

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 Class, and the return type for the return value of somfCreateIterator is a somf_TIterator Class. However, both methods return an instance of a somf_TDequeIterator that has been typed correctly.

Syntax

```
somf_TSequenceIterator
somfCreateSequenceIterator ();
```

Parameters

receiver A pointer to an object of class somf_TDeque.

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

Restrictions and Limitations

None.

Returned Values

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

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. Also, it deallocates the storage that these objects might have owned; that is, the destructor function is called for each object in the collection.

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

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

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

Syntax

```
void  somfDeleteAll ();
```

Parameters

receiver A pointer to an object of class somf_TDeque.

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

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_TDeque dq;
Environment *ev;
ev = somGetGlobalEnvironment();
dq = somf_TDequeNew();
/* Add some objects to dq */
/* Remove all objects in dq AND DELETE ALL INSTANCES */
_somfDeleteAll(dq,ev);
_somFree (dq);
```

Original Class

somf_TCollection

Related Information

None.

somfFirst Method

Purpose

Gets the first object in a deque collection.

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

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

Syntax

somf_MCollectible **somfFirst** ();

Parameters

receiver A pointer to an object of class somf_TDeque.

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

Restrictions and Limitations

None.

Returned Values

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

Related Information

somfAddAfter
somfBefore
somfLast

somfInsert Method

Purpose

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

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

Syntax

```
void  somfInsert(
    in somf_MCollectible 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 added to the deque.

Restrictions and Limitations

None.

Returned Values

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

somfRemoveQ

somfLast Method

Purpose

Gets the last object in a given deque collection.

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);
```

Syntax

somf_MCollectible **somfLast**();

Parameters

receiver A pointer to an object of class somf_TDeque.

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

Restrictions and Limitations

None.

Returned Values

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

Related Information

somfAfter
somfBefore
somfFirst

somfMember Method

Purpose

Gets an object in a deque collection.

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

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

Syntax

```
somf_MCollectible  somfMember(
    in somf_MCollectible 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.

Restrictions and Limitations

None.

Returned Values

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

Related Information

None.

somfPop Method

Purpose

Removes the object on top of a deque/stack.

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

Syntax

```
somf_MCollectible  somfPop ();
```

Parameters

receiver A pointer to an object of class somf_TDeque.

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

Restrictions and Limitations

None.

Returned Values

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

Example

```
somf_TDeque dq;
somf_MCollectible obj;
Environment *ev;
ev = somGetGlobalEnvironment();
dq = somf_TDequeNew();
/* Use _somfPush to push objects onto the stack */
/* Pop an object from the stack */
obj = _somfPop(dq,ev);
_somFree (dq);
```

Original Class

somf_TDeque

Related Information

somfPush

somfPush Method

Purpose

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

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

Syntax

```
void  somfPush(
    in somf_MCollectible 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 added to the deque.

Restrictions and Limitations

None.

Returned Values

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

somfPop

somfRemove Method

Purpose

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

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

Syntax

```
somf_MCollectible  somfRemove(
    in somf_MCollectible 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.

Restrictions and Limitations

None.

Returned Values

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

somfRemoveAll
somfRemoveFirst
somfRemoveLast

somfRemoveAll Method

Purpose

Removes all of the objects from a deque collection.

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

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

Syntax

```
void somfRemoveAll ();
```

Parameters

receiver A pointer to an object of class somf_TDeque.

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

Restrictions and Limitations

None.

Returned Values

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

Related Information

somfRemove
somfRemoveFirst
somfRemoveLast

somfRemoveFirst Method

Purpose

Removes the first object in a deque collection.

Syntax

```
somf_MCollectible  somfRemoveFirst ();
```

Parameters

receiver A pointer to an object of class somf_TDeque.

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

Restrictions and Limitations

None.

Returned Values

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

somfRemove
somfRemoveAll
somfRemoveLast

somfRemoveLast Method

Purpose

Removes the last object in a deque collection.

Syntax

```
somf_MCollectible  somfRemoveLast ();
```

Parameters

receiver A pointer to an object of class somf_TDeque.

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

Restrictions and Limitations

None.

Returned Values

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

somfRemove
somfRemoveAll
somfRemoveFirst

somfRemoveQ Method

Purpose

Removes the first object from a deque/queue.

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

Syntax

```
somf_MCollectible  somfRemoveQ ();
```

Parameters

receiver A pointer to an object of class somf_TDeque.

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

Restrictions and Limitations

None.

Returned Values

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

somfInsert

somfTDequeInitD Method

Purpose

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

Syntax

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

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.

Restrictions and Limitations

None.

Returned Values

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

somfTDequeInitF

somfTDequeInitF Method

Purpose

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

Syntax

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

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, residing in somf_MCollectible Class. The somf_TDeque object will use this pointer to access the somfIsSame Method or somfIsEqual Method that was declared and defined in the object being inserted into, or removed from, the somf_TDeque object.

Restrictions and Limitations

None.

Returned Values

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

somfTDequeInitD

Chapter 8. somf_TDequeIterator Class

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

Member Name:

tdeqitr

Import Name:

GOSSOMUC

Base Class:

somf_TSequenceIterator Class

Metaclass:

SOMClass

Ancestor Classes:

somf_TSequenceIterator
somf_TIterator
SOMObject

Subclasses:

somfTDequeIteratorInit

Types:

None.

Attributes:

None.

New Methods:

None.

Overriding Methods:

somfFirst
somfNext
somfLast
somfPrevious
somfRemove

somfFirst Method

Purpose

Resets the iterator and returns the first object from a deque collection. This resets the iterator to the beginning of the collection even if other operations on the collection cause the iterator to be invalidated. In the second case, this revalidates the iterator.

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

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

Syntax

somf_MCollectible **somfFirst** ();

Parameters

receiver A pointer to an object of class somf_TDequeIterator.

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

Restrictions and Limitations

None.

Returned Values

somf_MCollectible
A pointer to the first somf_MCollectible object in the collection.

SOMF_NIL Is returned if the collection is empty.

Example

```
somf_TDeque dq;
Environment *ev;
somf_TDequeIterator itr;
somf_MCollectible itrobj;
ev = somGetGlobalEnvironment();
dq = somf_TDequeNew();
itr = somf_TDequeIteratorNew();
_somfTDequeIteratorInit(itr, ev, dq);
/* Add some object to dq */
/* Iterate through the TDeque */
itrobj = somf_TDequeIterator_somfFirst(itr,ev);
while (itrobj != SOMF_NIL)
{
    /* Do something with itrobj */
    itrobj = _somfNext(itr,ev);
}
_somFree (dq);
_somFree (itr);
```

Original Class

somf_TIterator

Related Information

somfNext

somfLast Method

Purpose

Gets the last object in the deque collection.

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

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

Syntax

somf_MCollectible **somfLast** ();

Parameters

receiver A pointer to an object of class somf_TDequeIterator.

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

Restrictions and Limitations

None.

Returned Values

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

somfPrevious

somfNext Method

Purpose

Gets the next object in a deque collection. The method also returns a pointer to the next object, if found. Objects are retrieved in an order that reflects the "ordered-ness" of the collection, or the lack of ordering on the collection objects.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Thus, if any child of somf_TIterator is used with somf_TPrimitiveLinkedListIterator, then the name of the method must be fully qualified. This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you can reference this method as:

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

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

Syntax

```
somf_MCollectible  somfNext ();
```

Parameters

receiver A pointer to an object of class somf_TDequeIterator.

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

Restrictions and Limitations

None.

Returned Values

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

SOMF_NIL The end of the collection has been reached.

Example

```
somf_TDeque dq;
Environment *ev;
somf_TDequeIterator itr;
somf_MCollectible itrobj;
ev = somGetGlobalEnvironment();
dq = somf_TDequeNew();
itr = somf_TDequeIteratorNew();
_somfTDequeIteratorInit(itr, ev, dq);
/* Add some object to dq */
/* Iterate through the TDeque */
itrobj = somf_TDequeIterator_somfFirst(itr,ev);
while (itrobj != SOMF_NIL)
{
    /* Do something with itrobj */
    itrobj = _somfNext(itr,ev);
}
_somFree (dq);
_somFree (itr);
```

Original Class

somf_TIterator

Related Information

somfFirst

somfPrevious Method

Purpose

Gets the previous object in a deque collection. The method returns a pointer to the previous object, if found.

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

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

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

Syntax

somf_MCollectible **somfPrevious** ();

Parameters

receiver A pointer to an object of class somf_TDequeIterator.

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

Restrictions and Limitations

None.

Returned Values

somf_MCollectible
A pointer to the previous somf_MCollectible Class 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 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

somfLast

somfRemove Method

Purpose

Removes the current object from a deque collection. somfRemove is the only way to remove an object from a collection during iteration. If multiple iterators are in process, all the other iterators are invalidated.

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

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

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

Syntax

void **somfRemove** ();

Parameters

receiver A pointer to an object of class somf_TDequeIterator.

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

Restrictions and Limitations

None.

Returned Values

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

Related Information

None.

somfTDequeIteratorInit Method

Purpose

Initializes a somf_TDequeIterator for a deque collection.

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, and the final way is to use the somf_TDeque class's somfCreateSequenceIterator Method.

Note: You cannot override this method

Syntax

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

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.

Restrictions and Limitations

None.

Returned Values

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

Related Information

None.

Chapter 9. somf_TDequeLinkable Class

The somf_TDequeLinkable class is a subclass of somf_MLinkable Class. It provides a generic somf_MLinkable class to contain somf_MCollectible Class. 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 (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:

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

This class is not thread-safe.

This class is reentrant.

Member Name:

tdeqlink

Import Name:

GOSSOMUC

Base Class:

somf_MLinkable

Metaclass:

SOMClass

Ancestor Classes:

somf_MLinkable Class
SOMObject

Subclasses:

None.

Types:

None.

Attributes:

None.

New Methods:

somfGetValue
somfSetValue
somfTDequeLinkableInitDDM
somfTDequeLinkableInitDD

Overriding Methods:

somDefaultInit

somfGetValue Method

Purpose

Gets the value from a somf_TDequeLinkable node. The method returns a pointer to a somf_MCollectible Class object containing the value.

Syntax

```
somf_MCollectible  somfGetValue ();
```

Parameters

receiver A pointer to an object of class somf_TDequeLinkable.

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

Restrictions and Limitations

None.

Returned Values

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

Example

```
somf_TDequeLinkable dl;
<Your Class which inherits from somf_MCollectible> obj;
<Your Class which inherits from somf_MCollectible> obj2;
Environment *ev;
ev = somGetGlobalEnvironment();
obj = <Your Class which inherits from somf_MCollectible>New();
dl = somf_TDequeLinkableNew();
_somfSetValue(dl, ev, obj);
 obj2 = (<Your Class which inherits from somf_MCollectible>*)
          _somfGetValue(dl,ev);
_somFree (dl);
_somFree (obj);
```

Original Class

somf_TDequeLinkable

Related Information

somfSetValue

somfSetValue Method

Purpose

Sets the v alue of a given somf_TDequeLinkable node.

Syntax

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

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.

Restrictions and Limitations

None.

Returned Values

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

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. 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.

Syntax

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

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.

Restrictions and Limitations

None.

Returned Values

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

somfTDequeLinkableInitDDM

somfTDequeLinkableInitDDM Method

Purpose

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

Syntax

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

Parameters

receiver A pointer to an object of class somf_TDequeLinkable.

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

previous A pointer to the somf_TDequeLinkable before this one.

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

Restrictions and Limitations

None.

Returned Values

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

Example

```
somf_TDequeLinkable dl2;
Environment *ev;
ev = somGetGlobalEnvironment();
dl2 = somf_TDequeLinkableNew();
_somfTDequeLinkableInitDDM(dl2, ev, SOMF_NIL, SOMF_NIL,SOMF_NIL);
_somFree (dl2);
```

Original Class

somf_TDequeLinkable

Related Information

somfTDequeLinkableInitDD

Chapter 10. somf_TDictionary Class

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

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

The somf_TDictionary class is very similar to somf_THashTable. The primary difference is that somf_TDictionary inherits from somf_TCollection Class, whereas somf_THashTable does not. The other distinction is that the somf_TDictionary class uses the somfIsEqual Method as its default comparison function, whereas somf_THashTable uses somfIsSame Method. The somf_TDictionary class's use of somfIsEqual means that equal keys can only appear in the dictionary once.

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

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

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

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

Member Name:

tdict

Import Name:

GOSSOMUC

Base Class:

somf_TCollection

Metaclass:

SOMClass

Ancestor Classes:

somf_TCollection
somf_MCollectible
SOMObject

Subclasses:

None.

Types:

None.

Attributes:

None.

New Methods:

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

Overriding Methods:

somDefaultInit
somDestruct
somfAdd
somfCount
somfCreateIterator
somfDeleteAll
somfMember
somfRemove
somfRemoveAll

somfAdd Method

Purpose

Adds a specified *obj* to the dictionary. Do not be misled by this method's interface, which is inherited from the somf_TCollection Class. The only objects you can add with somfAdd are pairs of the somf_TAssoc Class. You cannot use this interface to add a generic somf_MCollectible Class object.

Syntax

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

Parameters

receiver A pointer to an object of class somf_TDictionary.

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

obj A pointer to an object that inherits from somf_MCollectible that will be added to the receiving object.

Restrictions and Limitations

None.

Returned Values

somf_MCollectible, a pointer to the somf_MCollectible object that was added, provided a duplicate object does not exist. Otherwise, it returns a pointer to the value of the duplicate object, if obj could not be added because a duplicate is already in the collection.

Example

```
somf_TDictionary d;
<Your Class which inherits from somf_MCollectible> obj;
<Your Class which inherits from somf_MCollectible> obj2;
somf_TAssoc tassoc;
Environment *ev;
ev = somGetGlobalEnvironment();
obj = <Your Class which inherits from somf_MCollectible>New();
obj2 = <Your Class which inherits from somf_MCollectible>New();
d = somf_TDictionaryNew();
tassoc = somf_TAssocNew();
_somfSetKey(tassoc, ev, obj);
_somfSetValue(tassoc, ev, obj2);
_somfAdd(d, ev, tassoc);
_somFree (d);
_somFree (obj);
_somFree (obj2);
_somFree (tassoc);
```

Original Class

somf_TCollection (overridden here)

Related Information

somfAddKeyValuePairMM
somfAddKeyValuePairMMB

somfAddKeyValuePairMM Method

Purpose

Adds a pair to the receiving dictionary object, and returns a removed object, if removal was necessary. The method also returns the value of the conflicting object (if any) that existed in the dictionary before this call.

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

Syntax

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

Parameters

receiver A pointer to an object of class somf_TDictionary.

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

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

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

Restrictions and Limitations

None.

Returned Values

somf_MCollectible

A pointer to the somf_MCollectible value that had to be removed in order for a new pair to be added.

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

Example

```
somf_TDictionary d;
<Your Class which inherits from somf_MCollectible> obj;
<Your Class which inherits from somf_MCollectible> obj2;
Environment *ev;
ev = somGetGlobalEnvironment();
obj = <Your Class which inherits from somf_MCollectible>New();
obj2 = <Your Class which inherits from somf_MCollectible>New();
d = somf_TDictionaryNew();
if (_somfAddKeyValuePairMM(d, ev, obj, obj2) != SOMF_NIL)
    somPrintf("\n problem adding obj,obj2 to d\n");
_somFree (d);
_somFree (obj);
_somFree (obj2);
```

Original Class

somf_TDictionary

Related Information

somfAdd
somfAddKeyValuePairMMB

somfAddKeyValuePairMMB Method

Purpose

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

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

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

Using the specified key and val arguments, this method transparently creates an object of the somf_TAssoc Class, before adding the new pair to the dictionary.

Syntax

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

Parameters

receiver A pointer to an object of class somf_TDictionary.

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

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

val A pointer to a somf_MCollectible object that is the value of the associated pair.

replace A boolean that indicates if an existing pair with an identical key should be replaced.

Restrictions and Limitations

None.

Returned Values

somf_MCollectible
A pointer to the somf_MCollectible value that had to be removed in order for a new pair to be added.

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

Example

```
somf_TDictionary d;
<Your Class which inherits from somf_MCollectible> obj2;
<Your Class which inherits from somf_MCollectible> obj3;
Environment *ev;
ev = somGetGlobalEnvironment();
obj2 = <Your Class which inherits from somf_MCollectible>New();
obj3 = <Your Class which inherits from somf_MCollectible>New();
d = somf_TDictionaryNew();
if (_somfAddKeyValuePairMMB(d, ev, obj2, obj3, TRUE)
   != SOMF_NIL)
    somPrintf("\n problem adding obj2,obj3 to d\n");
_somFree (d);
_somFree (obj2);
_somFree (obj3);
```

Original Class

somf_TDictionary

Related Information

somfAdd
somfAddKeyValuePairMM

somfAssign Method

Purpose

Assigns a dictionary as being "equal" to a given source dictionary. The method sets/resets the instance variables of the receiver to the values of the source. This operation is logically equivalent to using the equal (=) operator.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Thus, if any child of somf_TDictionary is used with any other main collection class, then the name of the method must be fully qualified. This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

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

Syntax

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

Parameters

receiver A pointer to an object of class somf_TDictionary.

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

source A pointer to the somf_TDictionary object to which the receiv ing object will be set equal.

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_TDictionary d;
somf_TDictionary d2;
Environment *ev;
ev = somGetGlobalEnvironment();
d = somf_TDictionaryNew();
d2 = somf_TDictionaryNew();
/* Add some objects to d */
/* Assign d2 = d */
somf_TDictionary_somfAssign(d2,ev,d);
_somFree (d);
_somFree (d2);
```

Original Class

somf_TDictionary

somfCopyImplementation Method

Purpose

Returns a hash table that is a copy of the hash table in a given dictionary. Normally, a client program will not need to invoke this method.

Syntax

```
somf_THashTable
somfCopyImplementation ();
```

Parameters

receiver A pointer to an object of class somf_TDictionary.

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

Restrictions and Limitations

None.

Returned Values

None.

Returned Values

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

Example

None.

Original Class

somf_TDictionary

Related Information

None.

somfCount Method

Purpose

Gets the number of objects in a dictionary.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Since somf_TDictionary uses somf_THashTable, the name of the method must be fully qualified. This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

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

Syntax

```
long somfCount ();
```

Parameters

receiver A pointer to an object of class somf_TDictionary.

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

Restrictions and Limitations

None.

Returned Values

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

Example

```
somf_TDictionary d;
<Your Class which inherits from somf_MCollectible> obj;
Environment *ev;
ev = somGetGlobalEnvironment();
d = somf_TDictionaryNew();
obj = <Your Class which inherits from somf_MCollectible>New();
somPrintf("\n Count of d= %d\n", somf_TDictionary_somfCount(d,ev));
_somFree (d);
_somFree (obj);
```

Original Class

somf_TCollection (overridden here)

Related Information

None.

somfCreateIterator Method

Purpose

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

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

Syntax

somf_TIterator **somfCreateIterator** ();

Parameters

receiver A pointer to an object of class somf_TDictionary.

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

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to the new iterator.

Example

```
somf_TDictionary d;
Environment *ev;
somf_TDictionaryIterator itr;
ev = somGetGlobalEnvironment();
d = somf_TDictionaryNew();
itr = (somf_TDictionaryIterator*) _somfCreateIterator(d,ev);
_somFree (d);
_somFree (itr);
```

Original Class

somf_TCollection (overridden here)

Related Information

None.

somfCreateNewImplementationF Method

Purpose

Creates a new hash table for a dictionary, given its comparison test method. The method also includes an argument that defines the comparison test method applied to current/potential dictionary objects.

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

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

Syntax

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

Parameters

receiver A pointer to an object of class somf_TDictionary.

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

testfn A method pointer specifying either a somfIsEqual or a 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 Class. The somf_TDictionary object will use this pointer to access the somfIsSame Method or somfIsEqual Method that was declared and defined in the object being inserted into, or removed from, the somf_TDictionary object.

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to the new hash table.

Example

```
somf_THashTable ht2;
somf_TDictionary d;
Environment *ev;
ev = somGetGlobalEnvironment();
d = somf_TDictionaryNew();
ht2 = _somfCreateNewImplementationF
       (d, ev, somf_MCollectibleClassData.somfIsEqual);
_somFree (d);
_somFree (ht2);
```

Original Class

somf_TDictionary

Related Information

somfCreateNewImplementationFL
somfCreateNewImplementationFLL
somfCreateNewImplementationFLLL

somfCreateNewImplementationFL Method

Purpose

Creates a new hash table for a dictionary, given its comparison test method and its initial table size. The method includes arguments that define the comparison test method applied to current/potential dictionary objects, and the initial size of the hash table.

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

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

Syntax

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

Parameters

receiver A pointer to an object of class somf_TDictionary.

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

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

This argument should always be set to either

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

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

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

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to the new hash table.

Example

```
somf_THashTable ht3;
somf_TDictionary d;
Environment *ev;
ev = somGetGlobalEnvironment();
d = somf_TDictionaryNew();
ht3 = _somfCreateNewImplementationFL(
      d, ev, somf_MCollectibleClassData.somfIsEqual, 23);
_somFree (d);
_somFree (ht3);
```

Original Class

somf_TDictionary

Related Information

somfCreateNewImplementationF
somfCreateNewImplementationFLL
somfCreateNewImplementationFLLL

somfCreateNewImplementationFLL Method

Purpose

Creates a new hash table for a dictionary, given its comparison test method, the initial table size, and the table's growth rate. The method includes arguments that define the comparison test method applied to current/potential dictionary objects, the initial table size, and the table's growth rate.

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

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

Syntax

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

Parameters

receiver A pointer to an object of class somf_TDictionary.

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

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

This argument should always be set to either

`somf_MCollectibleClassData.somfIsSame or somf_MCollectibleClassData.somfIsEqual`

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

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

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

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to the new hash table.

Example

```
somf_THashTable ht4;
somf_TDictionary d;
Environment *ev;
ev = somGetGlobalEnvironment();
d = somf_TDictionaryNew();
ht4 = _somfCreateNewImplementationFLL(
        d, ev, somf_MCollectibleClassData.somfIsEqual, 23, 20);
_somFree (d);
_somFree (ht4);
```

Original Class

somf_TDictionary

Related Information

somfCreateNewImplementationF
somfCreateNewImplementationFL
somfCreateNewImplementationFLLL

somfCreateNewImplementationFLLL Method

Purpose

Creates a new hash table for a dictionary, given its comparison test method, the initial table size, the table's growth rate, and the table's rehash threshold. The hash table is fully specified by arguments that define the comparison test method applied to current/potential dictionary objects, the initial table size, the table's growth rate, and the table's rehash threshold.

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

Syntax

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

Parameters

receiver A pointer to an object of class somf_TDictionary.

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

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

This argument should always be set to either

`somf_MCollectibleClassData.somfIsSame or somf_MCollectibleClassData.somfIsEqual`

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

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

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

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

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to the new hash table.

Example

```
somf_THashTable ht1;
somf_TDictionary d;
Environment *ev;
ev = somGetGlobalEnvironment();
d = somf_TDictionaryNew();
ht1 = _somfCreateNewImplementationFLLL(
      d, ev, somf_MCollectibleClassData.somfIsEqual,
      23, 20, 80);
_somFree (d);
_somFree (ht1);
```

Original Class

somf_TDictionary

Related Information

somfCreateNewImplementationF
somfCreateNewImplementationFL
somfCreateNewImplementationFLL

somfDeleteAll Method

Purpose

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

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

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Since somf_TDictionary uses somf_THashTable, then the name of this method must be fully qualified. This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

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

Syntax

void **somfDeleteAll** ();

Parameters

receiver A pointer to an object of class somf_TDictionary.

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

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_TDictionary d;
Environment *ev;
ev = somGetGlobalEnvironment();
d = somf_TDictionaryNew();
/* Add some objects in the somf_TDictionary */
/* Remove all the objects AND DELETE THEM */
somf_TDictionary_somfDeleteAll(d,ev);
_somFree (d);
```

Original Class

somf_TCollection (overridden here)

Related Information

somfDeleteAllKeys
somfDeleteAllValues
somfDeleteKey

somfDeleteAllKeys Method

Purpose

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

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Since somf_TDictionary uses somf_THashTable, then the name of this method must be fully qualified. This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

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

Syntax

void **somfDeleteAllKeys** ();

Parameters

receiver A pointer to an object of class somf_TDictionary.

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

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_TDictionary d;
Environment *ev;
ev = somGetGlobalEnvironment();
d = somf_TDictionaryNew();
/* Add some objects in the somf_TDictionary */
/* Remove all the objects AND DELETE ALL THE KEYS */
somf_TDictionary_somfDeleteAllKeys(d,ev);
_somFree (d);
```

Original Class

somf_TDictionary

Related Information

somfDeleteAll
somfDeleteAllValues
somfDeleteKey

somfDeleteAllValues Method

Purpose

Removes all of the pairs from a dictionary. The procedure resets the count to zero and calls the destructor on every value in the hash table. However, the program still owns the objects representing the keys of the pairs.

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

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Since somf_TDictionary uses somf_THashTable, the name of this method will have to be fully qualified. This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

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

Syntax

void **somfDeleteAllValues** ();

Parameters

receiver A pointer to an object of class somf_TDictionary.

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

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_TDictionary d;
Environment *ev;
ev = somGetGlobalEnvironment();
d = somf_TDictionaryNew();
/* Add some objects in the somf_TDictionary */
/* Remove all the objects AND DELETE ALL THE VALUES */
somf_TDictionary_somfDeleteAllValues(d,ev);
_somFree (d);
```

Original Class

somf_TDictionary

Related Information

somfDeleteAll
somfDeleteAllKeys
somfDeleteKey

somfDeleteKey Method

Purpose

Deletes a specified key from the associated pair, and removes the pair from the dictionary. The method also removes the pair from the dictionary represented by the receiving object. The method returns a pointer to the value from the pair.

Syntax

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

Parameters

receiver A pointer to an object of class somf_TDictionary.

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

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

Restrictions and Limitations

None.

Returned Values

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

SOMF_NIL The key object was not found.

Example

```
somf_TDictionary d;
<Your Class which inherits from somf_MCollectible> obj;
Environment *ev;
ev = somGetGlobalEnvironment();
obj = <Your Class which inherits from somf_MCollectible>New();
d = somf_TDictionaryNew();
/* Remove the key,value pair AND DELETE THE KEY */
if (_somfDeleteKey(d, ev, obj) == SOMF_NIL)
    somPrintf(" Why did DeleteKey say obj wasn't in d? \n");
_somFree (d);
```

Original Class

somf_TDictionary

Related Information

somfDeleteAll
somfDeleteAllKeys
somfDeleteAllValues

somfGetHashFunction Method

Purpose

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

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Since somf_TDictionary uses somf_THashTable, the name of this method will have to be fully qualified. This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

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

Syntax

somf_MCollectibleHashFn
somfGetHashFunction ();

Parameters

receiver A pointer to an object of class somf_TDictionary.

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

Restrictions and Limitations

None.

Returned Values

somf_MCollectible

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

Example

```
somf_TDictionary d;
Environment *ev;
ev = somGetGlobalEnvironment();
d = somf_TDictionaryNew();
if ((somf_TDictionary_somfGetHashFunction(d,ev)) !=
              somf_MCollectibleClassData.somfHash)
    somPrintf("\n What Hash Function are we using?\n");
_somFree (d);
```

Original Class

somf_TDictionary

Related Information

somfSetHashFunction

somfKeyAtM Method

Purpose

Gets a dictionary's first key that has a specified val as its associated value.

Note: This method involves a slow search.

Syntax

```
somf_MCollectible  somfKeyAtM(
    in somf_MCollectible val);
```

Parameters

receiver A pointer to an object of class somf_TDictionary.

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

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

Restrictions and Limitations

None.

Returned Values

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

SOMF_NIL The value was not found in the dictionary.

Example

```
somf_TDictionary d;
<Your Class which inherits from somf_MCollectible> key;
<Your Class which inherits from somf_MCollectible> value;
Environment *ev;
ev = somGetGlobalEnvironment();
d = somf_TDictionaryNew();
value = <Your Class which inherits from somf_MCollectible>New();
/* Add a lot of objects to d */
key = _somfKeyAtM(d, ev, value);
if (key == SOMF_NIL)
   somPrintf(" value is no longer in d\n");
else
{
    /* do something with the key */
}
_somFree (d);
_somFree (value);
```

Original Class

somf_TDictionary

Related Information

somfKeyAtMF

somfKeyAtMF Method

Purpose

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

Note: This method involves a slow search.

Syntax

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

Parameters

receiver A pointer to an object of class somf_TDictionary.

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

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

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

This argument should always be set to either:

`somf_MCollectibleClassData.somfIsSame or somf_MCollectibleClassData.somfIsEqual.`

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

Restrictions and Limitations

None.

Returned Values

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

SOMF_NIL The value was not found in the dictionary.

Example

```
somf_TDictionary d;
<Your Class which inherits from somf_MCollectible> key;
<Your Class which inherits from somf_MCollectible> value;
Environment *ev;
ev = somGetGlobalEnvironment();
d = somf_TDictionaryNew();
value = <Your Class which inherits from somf_MCollectible>New();
/* Add a lot of objects to d */
key = _somfKeyAtMF(d, ev, value,
                   somf_MCollectibleClassData.somfIsEqual)
if (key == SOMF_NIL)
    somPrintf(" value is no longer in d\n");
else
{
    /* do something with the key */
}
_somFree (d);
_somFree (value);
```

Original Class

somf_TDictionary

Related Information

somfKeyAtM

somfMember Method

Purpose

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

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Since somf_TDictionary uses somf_THashTable, then the name of the method will have to be fully qualified. This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

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

Syntax

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

Parameters

receiver A pointer to an object of class somf_TDictionary.

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

key A pointer to the somf_MCollectible Class key of the (key, value) pair that may or may not be in the dictionary.

Restrictions and Limitations

None.

Returned Values

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

SOMF_NIL The object was not found.

Example

```
somf_TDictionary d;
<Your Class which inherits from somf_MCollectible> obj;
Environment *ev;
ev = somGetGlobalEnvironment();
d = somf_TDictionaryNew();
obj = <Your Class which inherits from somf_MCollectible>New();
/* Add some objects to d */
/* See if obj is in d */
if (somf_TDictionary_somfMember(d, ev, obj) == SOMF_NIL)
    somPrintf("\n obj is NOT in d\n");
else
    somPrintf("\n obj IS in d\n");
_somFree (d);
```

Original Class

somf_TCollection (overridden here)

Related Information

None.

somfRemove Method

Purpose

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

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Since somf_TDictionary uses somf_THashTable, then the name of the method will have to be fully qualified. This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

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

Syntax

```
somf_MCollectible  somfRemove(
    in somf_MCollectible key);
```

Parameters

receiver A pointer to an object of class somf_TDictionary.

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

key A pointer to the somf_MCollectible Class object representing the key to be removed from the dictionary.

Restrictions and Limitations

None.

Returned Values

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

SOMF_NIL The key object was not found

Example

```
somf_TDictionary d;
<Your Class which inherits from somf_MCollectible> key;
Environment *ev;
ev = somGetGlobalEnvironment();
d = somf_TDictionaryNew();
key = <Your Class which inherits from somf_MCollectible>New();
/* Add a lot of objects to d */
if (somf_TDictionary_somfRemove(d, ev, key) == SOMF_NIL)
    somPrintf(" Why did Remove say key was not removed?\n");
_somFree (d);
_somFree (key);
```

Original Class

somf_TCollection (overridden here)

Related Information

somfRemoveAll

somfRemoveAll Method

Purpose

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

C cannot handle methods from different classes having the same name when the inherit the name from different parents. Since somf_TDictionary uses somf_THashTable, the name of the method will have to be fully qualified. This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

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

Syntax

```
void  somfRemoveAll ();
```

Parameters

receiver A pointer to an object of class somf_TDictionary.

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

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_TDictionary d;
Environment *ev;
ev = somGetGlobalEnvironment();
d = somf_TDictionaryNew();
/* Add a lot of objects to d */
/* remove all the objects from d */
somf_TDictionary_somfRemoveAll(d,ev);
_somFree (d);
```

Original Class

somf_TCollection (overridden here)

Related Information

somfRemove

somfSetHashFunction Method

Purpose

Sets a dictionary's hash-function pointer to a given method. By default, this pointer is set to somf_MCollectible's somfHash Method, which is usually overridden in the objects that are added to the hash table. Normally, a client program does not invoke this method.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Since somf_TDictionary uses somf_THashTable, the name of this method will have to be fully qualified. This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

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

Syntax

```
void  somfSetHashFunction(
    in somf_MCollectibleHashFn fn);
```

Parameters

receiver A pointer to an object of class somf_TDictionary.

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

fn A method pointer specifying a somfHash type method.

This argument should always be set to

```
somf_MCollectibleClassData.somfHash
```

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

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_TDictionary d;
Environment *ev;
ev = somGetGlobalEnvironment();
d = somf_TDictionaryNew();
somf_TDictionary_somfSetHashFunction(d, ev,
                    somf_MCollectibleClassData.somfHash);
_somFree (d);
```

Original Class

somf_TDictionary

Related Information

somfGetHashFunction

somfTDictionaryInitD Method

Purpose

Initializes a new dictionary, setting it equal to another specified dictionary. The method also sets the new dictionary equal to another specified dictionary. This implies that the instance data of the new dictionary will be set equal to those of the source dictionary.

Note: You cannot override this method.

Syntax

```
somf_TDictionary
somfTDictionaryInitD(
    in somf_TDictionary dictionary);
```

Parameters

receiver A pointer to an object of class somf_TDictionary.

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

dictionary A pointer to the dictionary the receiving object will be equal to.

Restrictions and Limitations

None.

Returned Values

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

Example

```
somf_TDictionary d2;
somf_TDictionary d7;
Environment *ev;
ev = somGetGlobalEnvironment();
d2 = somf_TDictionaryNew();
d7 = somf_TDictionaryNew();
_somfTDictionaryInitD(d7, ev, d2);
_somFree (d2);
_somFree (d7);
```

Original Class

somf_TDictionary

Related Information

somfTDictionaryInitF
somfTDictionaryInitFL
somfTDictionaryInitFLL
somfTDictionaryInitL
somfTDictionaryInitLL
somfTDictionaryInitLLF

somfTDictionaryInitF Method

Purpose

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

Note: You cannot override this method.

Syntax

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

Parameters

receiver A pointer to an object of class somf_TDictionary.

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

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

This argument should always be set to either

`somf_MCollectibleClassData.somfIsSame or somf_MCollectibleClassData.somfIsEqual`

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

Restrictions and Limitations

None.

Returned Values

A pointer to an initialized somf_TDictionary object representing the new dictionary.

Example

```
somf_TDictionary d3;
Environment *ev;
ev = somGetGlobalEnvironment();
d3 = somf_TDictionaryNew();
_somfTDictionaryInitF(d3, ev,
                 somf_MCollectibleClassData.somfIsEqual);
_somFree (d3);
```

Original Class

somf_TDictionary

Related Information

None.

somfTDictionaryInitFL Method

Purpose

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

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

Note: You cannot override this method.

Syntax

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

Parameters

receiver A pointer to an object of class somf_TDictionary.

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

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

This argument should always be set to either

`somf_MCollectibleClassData.somfIsSame or somf_MCollectibleClassData.somfIsEqual.`

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

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

Restrictions and Limitations

None.

Returned Values

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

Example

```
somf_TDictionary d2;
Environment *ev;
ev = somGetGlobalEnvironment();
d2 = somf_TDictionaryNew();
_somfTDictionaryInitFL(d2, ev,
                somf_MCollectibleClassData.somfIsEqual, 8);
_somFree (d2);
```

Original Class

somf_TDictionary

Related Information

somfTDictionaryInitD
somfTDictionaryInitF
somfTDictionaryInitFLL
somfTDictionaryInitL
somfTDictionaryInitLL
somfTDictionaryInitLLF

somfTDictionaryInitFLL Method

Purpose

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

Note: This method is equivalent to the somfTDictionaryInitLLF Method.

Note: You cannot override this method.

Syntax

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

Parameters

receiver A pointer to an object of class somf_TDictionary.

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

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

This argument should always be set to either

`somf_MCollectibleClassData.somfIsSame or somf_MCollectibleClassData.somfIsEqual.`

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

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

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

Restrictions and Limitations

None.

Returned Values

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

Example

```
somf_TDictionary d1;
Environment *ev;
ev = somGetGlobalEnvironment();
d1 = somf_TDictionaryNew();
_somfTDictionaryInitFLL(d1, ev,
            somf_MCollectibleClassData.somfIsEqual, 8, 8);
_somFree (d1);
```

Original Class

somf_TDictionary

Related Information

somfTDictionaryInitD
somfTDictionaryInitF
somfTDictionaryInitFL
somfTDictionaryInitL
somfTDictionaryInitLL
somfTDictionaryInitLLF

somfTDictionaryInitL Method

Purpose

Initializes a new dictionary, given its initial size.

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

Note: You cannot override this method.

Syntax

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

Parameters

receiver A pointer to an object of class somf_TDictionary.

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

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

Restrictions and Limitations

None.

Returned Values

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

Example

```
somf_TDictionary d6;
Environment *ev;
ev = somGetGlobalEnvironment();
d6 = somf_TDictionaryNew();
_somfTDictionaryInitL(d6, ev, 8);
_somFree (d6);
```

Original Class

somf_TDictionary

Related Information

somfTDictionaryInitD
somfTDictionaryInitF
somfTDictionaryInitFL
somfTDictionaryInitFLL
somfTDictionaryInitLL
somfTDictionaryInitLLF

somfTDictionaryInitLL Method

Purpose

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

Note: You cannot override this method.

Syntax

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

Parameters

receiver A pointer to an object of class somf_TDictionary.

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

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

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

Restrictions and Limitations

None.

Returned Values

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

Example

```
somf_TDictionary d5;
Environment *ev;
ev = somGetGlobalEnvironment();
d5 = somf_TDictionaryNew();
_somfTDictionaryInitLL(d5, ev, 8, 8);
_somFree (d5);
```

Original Class

somf_TDictionary

Related Information

somfTDictionaryInitD
somfTDictionaryInitF
somfTDictionaryInitFL
somfTDictionaryInitFLL
somfTDictionaryInitL
somfTDictionaryInitLLF

somfTDictionaryInitLLF Method

Purpose

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

Note: This method is equivalent to the somfTDictionaryInitFLL Method. method.

Note: You cannot override this method.

Syntax

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

Parameters

receiver A pointer to an object of class somf_TDictionary.

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

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

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

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

This argument should always be set to either

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

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

Restrictions and Limitations

None.

Returned Values

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

Example

```
somf_TDictionary d4;
Environment *ev;
ev = somGetGlobalEnvironment();
d4 = somf_TDictionaryNew();
_somfTDictionaryInitLLF(d4, ev, 8, 8,
                  somf_MCollectibleClassData.somfIsEqual);
_somFree (d4);
```

Original Class

somf_TDictionary

Related Information

somfTDictionaryInitD
somfTDictionaryInitF
somfTDictionaryInitFL
somfTDictionaryInitFLL
somfTDictionaryInitL
somfTDictionaryInitLL

somfValueAt Method

Purpose

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

Syntax

```
somf_MCollectible  somfValueAt
    (in somf_MCollectible key);
```

Parameters

receiver A pointer to an object of class somf_TDictionary.

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

key A pointer to a somf_MCollectible Class object that is the key to be searched for.

Restrictions and Limitations

None.

Returned Values

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

SOMF_NIL The key was not found in the dictionary.

Example

```
somf_TDictionary d;
somf_MCollectible value;
<Your Class which inherits from somf_MCollectible> key;
Environment *ev;
ev = somGetGlobalEnvironment();
d = somf_TDictionaryNew();
key = <Your Class which inherits from somf_MCollectible>New();
/* Add a lot of objects to d */
value = _somfValueAt(d, ev, key);
_somFree (d);
_somFree (key);
```

Original Class

somf_TDictionary

Related Information

None.

Chapter 11. somf_TDictionaryIterator Class

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

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

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

Member Name:

tdictitr

Import Name:

GOSSOMUC

Base Class:

somf_TIterator

Metaclass:

SOMClass

Ancestor Classes:

somf_TIterator
SOMObject

Subclasses:

None.

Types:

None.

Attributes:

None.

New Methods:

somfTDictionaryIteratorInit

Overriding Methods:

somDefaultInit
somDestruct
somfFirst
somfNext
somfRemove

somfFirst Method

Purpose

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

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

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

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

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

Syntax

somf_MCollectible **somfFirst** ();

Parameters

receiver A pointer to an object of class somf_TDictionaryIterator.

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

Restrictions and Limitations

None.

Returned Values

somf_MCollectible
A pointer to the first somf_MCollectible object in the dictionary.

SOMF_NIL Is returned if the collection is empty.

Example

```
somf_TDictionary d;
Environment *ev;
somf_TDictionaryIterator itr;
somf_TAssoc itrobj;
somf_MCollectible objk;
somf_MCollectible objv;
ev = somGetGlobalEnvironment();
d = somf_TDictionaryNew();
itr = somf_TDictionaryIteratorNew();
_somfTDictionaryIteratorInit(itr, ev, d);
/* Add some object to d */
/* Iterate through the TDictionary */
itrobj = somf_TDictionaryIterator_somfFirst(itr,ev);
while (itrobj != SOMF_NIL)
{
    objk = _somfGetKey(itrobj,ev);
    objv = _somfGetValue(itrobj,ev);
   /* Do something with objk or objv */
    itrobj = _somfNext(itr,ev);
}
_somFree (d);
_somFree (itr);
```

Original Class

somf_TIterator (overridden here)

Related Information

somfNext

somfNext Method

Purpose

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

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

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

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Thus, if any child of somf_TIterator is used with an object of the somf_TPrimitiveLinkedListIterator class, then the name of the method must be fully qualified (example: somf_TDictionaryIterator_somfNext). This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

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

Syntax

somf_MCollectible **somfNext** ();

Parameters

receiver A pointer to an object of class somf_TDictionaryIterator.

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

Restrictions and Limitations

None.

Returned Values

somf_MCollectible
A pointer to the next somf_MCollectible object in the dictionary.

SOMF_NIL The end of the dictionary has been reached.

Example

```
somf_TDictionary d;
Environment *ev;
somf_TDictionaryIterator itr;
somf_TAssoc itrobj;
somf_MCollectible objk;
somf_MCollectible objv;
ev = somGetGlobalEnvironment();
d = somf_TDictionaryNew();
itr = somf_TDictionaryIteratorNew();
_somfTDictionaryIteratorInit(itr, ev, d);
/* Add some object to d */
/* Iterate through the TDictionary */
itrobj = somf_TDictionaryIterator_somfFirst(itr,ev);
while (itrobj != SOMF_NIL)
{
    objk = _somfGetKey(itrobj,ev);
    objv = _somfGetValue(obj,ev);
   /* Do something with objk or objv */
    itrobj = _somfNext(itr,ev);
}
_somFree (d);
_somFree (itr);
```

Original Class

somf_TIterator (overridden here)

Related Information

somfFirst

somfRemove Method

Purpose

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

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

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

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

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

Syntax

```
void  somfRemove ();
```

Parameters

receiver A pointer to an object of class somf_TDictionaryIterator.

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

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_TDictionary d;
Environment *ev;
somf_TDictionaryIterator itr;
somf_MCollectible itrobj;
ev = somGetGlobalEnvironment();
d = somfdictionaryinit();
itr = somf_TDictionaryIteratorNew();
_somfTDictionaryIteratorInit(itr, ev, d);
/* Add some objects to d */
/* Use the Iterator's Remove to remove the first object */
itrobj = somf_TDictionaryIterator_somfFirst(itr,ev);
somf_TDictionaryIterator_somfRemove(itr,ev);
_somFree (d);
_somFree (itr);
```

Original Class

somf_TIterator

Related Information

None.

somfTDictionaryIteratorInit Method

Purpose

Initializes a somf_TDictionaryIterator iterator for a specified dictionary.

This is one of two ways to initialize a somf_TDictionaryIterator to point to an instance of a somf_TDictionary dictionary. The other way is to use the somf_TDictionary class's somfCreateIterator Method.

Note: You cannot override this method.

Syntax

```
somf_TDictionaryIterator
somfTDictionaryIteratorInit(
    in somf_TDictionary h);
```

Parameters

receiver A pointer to an object of class somf_TDictionaryIterator.

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

h A pointer to the dictionary object that the receiving object will iterate over.

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to an initialized somf_TDictionaryIterator object.

Example

```
somf_TDictionary d;
Environment *ev;
somf_TDictionaryIterator itr;
ev = somGetGlobalEnvironment();
d = somf_TDictionaryNew();
itr = somf_TDictionaryIteratorNew();
_somfTDictionaryIteratorInit (itr, ev, d);
_somFree (d);
_somFree (itr);
```

Original Class

somf_TDictionaryIterator

Related Information

None.

Chapter 12. somf_THashTable Class

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

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

The somf_THashTable class is very similar to somf_TDictionary. The primary difference is that somf_THashTable inherits directly from the somf_MCollectible Class, whereas somf_TDictionary is another level down, inheriting from somf_TCollection Class. The other distinction is that the somf_THashTable class uses the somfIsSame Method as its default comparison function, whereas somf_TDictionary uses somfIsEqual Method.

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

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

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

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

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

Member Name:

thash

Import Name:

GOSSOMUC

Base Class:

somf_MCollectible

Metaclass:

SOMClass

Ancestor Classes:

somf_MCollectible
SOMObject

Subclasses:

None.

Types:

None.

Attributes:

None.

New Methods:

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

Overriding Methods:

somDefaultInit
somDestruct

somfAddMM Method

Purpose

Adds a pair to the hash table. This method will replace a copy if one already exists. Using the arguments, this method transparently creates an object of the somf_TAssoc Class, before adding the new pair to the hash table.

Note: You cannot override this method.

Syntax

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

Parameters

receiver A pointer to an object of class somf_THashTable.

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

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

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

Restrictions and Limitations

None.

Returned Values

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

SOMF_NIL No object had to be removed to add the new pair.

Example

```
somf_THashTable ht;
<Your Class which inherits from somf_MCollectible> obj;
<Your Class which inherits from somf_MCollectible> obj2;
Environment *ev;
ev = somGetGlobalEnvironment();
obj = <Your Class which inherits from somf_MCollectible>New();
obj2 = <Your Class which inherits from somf_MCollectible>New();
ht = somf_THashTableNew();
if (_somfAddMM(ht, ev, obj, obj2) != SOMF_NIL)
    somPrintf("\n problem adding obj,obj2 to ht\n");
_somFree (ht);
_somFree (obj);
_somFree (obj2);
```

Original Class

somf_THashTable

Related Information

somfAddMMB

somfAddMMB Method

Purpose

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

If *replace* = TRUE, the pair is added to the hash table regardless of whether a copy (a pair having the same key) already exists. Otherwise, if *replace* = FALSE, then the pair is added to the hash table only if a copy does not exist. Using the specified *key* and *value* arguments, this method transparently creates an object of the somf_TAssoc Class, before adding the new pair to the hash table.

Note: You cannot override this method.

Syntax

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

Parameters

receiver A pointer to an object of class somf_THashTable.

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

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

value A pointer to a somf_MCollectible object that is the value of the associated pair.

replace A boolean indicating whether an already existing pair with an identical key should be replaced.

Restrictions and Limitations

None.

Returned Values

somf_MCollectible
A pointer to thesomf_MCollectible value that had to be removed in order to add a new pair.

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

Example

```
somf_THashTable ht;
<Your Class which inherits from somf_MCollectible> obj2;
<Your Class which inherits from somf_MCollectible> obj3;
Environment *ev;
ev = somGetGlobalEnvironment();
obj2 = <Your Class which inherits from somf_MCollectible>New();
obj3 = <Your Class which inherits from somf_MCollectible>New();
ht = somf_THashTableNew();
if (_somfAddMMB(ht, ev, obj2, obj3, FALSE) != SOMF_NIL)
        somPrintf("\n problem adding obj2,obj3 to ht\n");
_somFree (ht);
_somFree (obj2);
_somFree (obj3);
```

Original Class

somf_THashTable

Related Information

somfAddMM

somfAssign Method

Purpose

Assigns a hash table as being equal to a given source hash table. The method sets/resets the instance variables of the receiver to the values of the source. This operation is logically equivalent to using the equal (=) operator.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Thus, if any child of somf_THashTable is used with any other main collection class, then the name of the method must be fully qualified. This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

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

Syntax

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

Parameters

receiver A pointer to an object of class somf_THashTable.

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

source A pointer to the somf_THashTable to which the receiving object should be set equal.

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_THashTable h1;
somf_THashTable h2;
Environment *ev;
ev = somGetGlobalEnvironment();
h1 = somf_THashTableNew();
h2 = somf_THashTableNew();
/* Add a lot of objects to h1 */
/* Assign h2 to the contents of h1 */
somf_THashTable_somfAssign(h2,ev,h1);
_somFree (h1);
_somFree (h2);
```

Original Class

somf_THashTable

Related Information

None.

somfCount Method

Purpose

Gets the number of objects in the hash table.

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

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

Note: You cannot override this method.

Syntax

long **somfCount** ();

Parameters

receiver A pointer to an object of class somf_THashTable.

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

Restrictions and Limitations

None.

Returned Values

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

Example

```
somf_THashTable ht;
Environment *ev;
ev = somGetGlobalEnvironment();
ht = somf_THashTableNew();
/* Add some objects to ht */
/* Print the number of objects in ht */
somPrintf("\n Count of ht= %d\n",
          somf_THashTable_somfCount(ht,ev));
_somFree (ht)
```

Original Class

somf_THashTable

Related Information

None.

somfDelete Method

Purpose

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

A hash table does not contain copies of the objects, but pointers to the objects. Using somfDelete deletes the original object. If an attempt is made to delete a somf_MCollectible Class object already deleted, this will cause a segmentation violation.

Note: You cannot override this method.

Syntax

```
somf_MCollectible  somfDelete(
    in somf_MCollectible key);
```

Parameters

receiver A pointer to an object of class somf_THashTable.

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

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

Restrictions and Limitations

None.

Returned Values

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

SOMF_NIL The key object was not found.

Example

```
somf_THashTable ht;
<Your Class which inherits from somf_MCollectible> obj;
Environment *ev;
ev = somGetGlobalEnvironment();
obj = <Your Class which inherits from somf_MCollectible>New();
ht = somf_THashTableNew();
/* Add a lot of objects to ht */
/* Remove all occurrences of obj from ht AND DELETE obj */
if (_somfDelete(ht, ev, obj) == SOMF_NIL)
    somPrintf(" Why did Delete say obj wasn't in ht? \n");
_somFree (ht);
```

Original Class

somf_THashTable

Related Information

somfDeleteAll
somfDeleteAllKeys
somfDeleteAllValues

somfDeleteAll Method

Purpose

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

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

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

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

Note: You cannot override this method.

Syntax

```
void somfDeleteAll ();
```

Parameters

receiver A pointer to an object of class somf_THashTable.

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

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_THashTable ht;
Environment *ev;
ev = somGetGlobalEnvironment();
ht = somf_THashTableNew();
/* Add a lot of objects to ht */
/* Remove all objects from ht AND DELETE THEM */
somf_THashTable_somfDeleteAll(ht,ev);
_somFree (ht);
```

Original Class

somf_THashTable

Related Information

somfDelete
somfDeleteAllKeys
somfDeleteAllValues

somfDeleteAllKeys Method

Purpose

Removes all of the pairs from a hash-table receiving object. However, the program still owns the values of the pairs. However, the program still owns the values of the pairs. The destructor function is called for each key in the hash table. However, the program still owns the objects representing the values of the pairs.

Be careful with somfDeleteAllKeys. A hash table does not contain copies of the objects; it contains pointers to the objects. Using somfDeleteAllKeys deletes the original object. If an attempt is made to delete a somf_MCollectible Class object that has already been deleted, this will cause a segmentation violation.

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

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

Note: You cannot override this method.

Syntax

void **somfDeleteAllKeys** ();

Parameters

receiver A pointer to an object of class somf_THashTable.

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

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_THashTable ht;
Environment *ev;
ev = somGetGlobalEnvironment();
ht = somf_THashTableNew();
/* Add a lot of objects to ht */
/* Remove all objects from ht AND DELETE ALL THE KEYS */
somf_THashTable_somfDeleteAllKeys(ht,ev);
_somFree (ht);
```

Original Class

somf_THashTable

Related Information

somfDeleteAll
somfDelete
somfDeleteAllValues

somfDeleteAllValues Method

Purpose

Removes all of the pairs from a hash table. However, the program still owns the keys of the pairs. It also deallocates the storage that these objects might have owned and resets the count to zero The destructor function is called for each value in the hash table. However, the program still owns the objects representing the keys of the pairs.

Be careful with somfDeleteAllValues. A hash table does not contain copies of the objects; it contains pointers to the objects. Using somfDeleteAllValues deletes the original object. If an attempt is made to delete a somf_MCollectible Class object that has already been deleted, this will cause a segmentation violation.

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

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

Note: You cannot override this method.

Syntax

void **somfDeleteAllValues** ();

Parameters

receiver A pointer to an object of class somf_THashTable.

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

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_THashTable ht;
Environment *ev;
ev = somGetGlobalEnvironment();
ht = somf_THashTableNew();
/* Add a lot of objects to ht */
/* Remove all objects from ht AND DELETE ALL THE VALUES */
somf_THashTable_somfDeleteAllValues(ht,ev);
_somFree (ht);
```

Original Class

somf_THashTable

Related Information

somfDeleteAll
somfDeleteAllKeys
somfDelete

somfGetGrowthRate Method

Purpose

Gets the growth rate of a given hash table.

Note: You cannot override this method.

Syntax

```
long somfGetGrowthRate ();
```

Parameters

receiver A pointer to an object of class somf_THashTable.

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

Restrictions and Limitations

None.

Returned Values

This method returns the growth rate for the hash table.

Example

```
somf_THashTable ht;
Environment *ev;
ev = somGetGlobalEnvironment();
ht = somf_THashTableNew();
somPrintf(" Rate should be 20 and is %d \n",
          _somfGetGrowthRate(ht,ev);
_somFree (ht);
```

Original Class

somf_THashTable

Related Information

somfSetGrowthRate

somfGetHashFunction Method

Purpose

Gets the hash function used by a given hash table.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Thus, if a child of somf_THashTable is used with a child of somf_TDictionary or somf_TSet, then the name of the method will have to be fully qualified. This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

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

Note: You cannot override this method.

Syntax

somf_MCollectibleHashFn
somfGetHashFunction ();

Parameters

receiver A pointer to an object of class somf_THashTable.

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

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to the hash function.

Example

```
somf_THashTable ht;
Environment *ev;
ev = somGetGlobalEnvironment();
ht = somf_THashTableNew();
if ((somf_THashTable_somfGetHashFunction(ht,ev)) !=
                 somf_MCollectibleClassData.somfHash)
    somPrintf("\n What Hash Function are we using?\n");
_somFree (ht);
```

Original Class

somf_THashTable

Related Information

somfSetHashFunction

somfGetRehashThreshold Method

Purpose

Gets the rehash threshold of a given hash table. The rehash threshold is the percentage of how full the hash table should be before it needs to grow. For example: 80 means 80% of the hash table should be full before the table needs to grow.

Note: You cannot override this method.

Syntax

```
long somfGetRehashThreshold ();
```

Parameters

receiver A pointer to an object of class somf_THashTable.

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

Restrictions and Limitations

None.

Returned Values

This method returns the rehash threshold of the hash table.

Example

```
somf_THashTable ht;
Environment *ev;
ev = somGetGlobalEnvironment();
ht = somf_THashTableNew();
somPrintf(" RehashThreshold should be 80 and is %d \n",
          _somfGetRehashThreshold(ht,ev));
_somFree (ht);
```

Original Class

somf_THashTable

Related Information

somfSetRehashThreshold

somfGrow Method

Purpose

Grows a given hash table.

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

Note: You cannot override this method.

Syntax

```
void  somfGrow ();
```

Parameters

receiver A pointer to an object of class somf_THashTable.

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

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_THashTable ht;
Environment *ev;
ev = somGetGlobalEnvironment();
ht = somf_THashTableNew();
/* Add a lot of objects to ht */
_somfGrow(ht,ev);
_somFree (ht);
```

Original Class

somf_THashTable

Related Information

None.

somfMember Method

Purpose

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

The somfMember method determines whether the pair corresponding to a specified *key* is in the hash table represented by the receiving object and, if so, returns a pointer to the key object.

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

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

Note: You cannot override this method.

Syntax

```
somf_MCollectible  somfMember(
    in somf_MCollectible key);
```

Parameters

receiver A pointer to an object of class somf_THashTable.

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

key A pointer to the somf_MCollectible key of the pair that may or may not be a member of the Hash Table.

Restrictions and Limitations

None.

Returned Values

somf_MCollectible
A pointer to the key of the pair the method determined as the member.

SOMF_NIL The object was not found.

Example

```
somf_THashTable ht;
<Your Class which inherits from somf_MCollectible> obj;
Environment *ev;
ev = somGetGlobalEnvironment();
obj = <Your Class which inherits from somf_MCollectible>New();
ht = somf_THashTableNew();
/* Add a lot of objects to ht */
/* See if obj is in ht */
if (somf_THashTable_somfMember(ht, ev,obj) != SOMF_NIL)
    somPrintf("\n obj IS in ht\n");
else
    somPrintf("\n obj is NOT in ht\n");
_somFree (ht);
```

Original Class

somf_THashTable

Related Information

None.

somfRemove Method

Purpose

Removes from the hash table the pair associated with a given key.

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

```
d->somfremoveall(ev, key);
```

Note: You cannot override this method.

Syntax

```
somf_MCollectible  somfRemove(
    in somf_MCollectible key);
```

Parameters

receiver A pointer to an object of class somf_THashTable.

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

key A pointer to the somf_MCollectible key of the pair to be removed.

Restrictions and Limitations

None.

Returned Values

somf_MCollectible
A pointer to the value of the pair that was removed.

SOMF_NIL The *key* object was not found.

Example

```
somf_THashTable ht;
<Your Class which inherits from somf_MCollectible> key;
Environment *ev;
ev = somGetGlobalEnvironment();
ht = somf_THashTableNew();
key = <Your Class which inherits from somf_MCollectible>New();
/* Add a lot of objects to ht */
if (somf_THashTable_somfRemove(ht, ev, key) == SOMF_NIL)
    somPrintf(" Why did Remove say key was not removed?\n");
_somFree (ht);
_somFree (key);
```

Original Class

somf_THashTable

Related Information

somfRemoveAll

somfRemoveAll Method

Purpose

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

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

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

Note: You cannot override this method.

Syntax

void **somfRemoveAll** ();

Parameters

receiver A pointer to an object of class somf_THashTable.

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

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_THashTable ht;
Environment *ev;
ev = somGetGlobalEnvironment();
ht = somf_THashTableNew();
/* Add a lot of objects to ht */
/* Remove all the objects from ht */
somf_THashTable_somfRemoveAll(ht,ev);
_somFree (ht);
```

Original Class

somf_THashTable

Related Information

somfRemove

somfRetrieve Method

Purpose

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

Note: You cannot override this method.

Syntax

```
somf_MCollectible  somfRetrieve(
    in somf_MCollectible key);
```

Parameters

receiver A pointer to an object of class somf_THashTable.

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

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

Restrictions and Limitations

None.

Returned Values

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

SOMF_NIL The key was not found in the hash table.

Example

```
somf_THashTable ht;
<Your Class which inherits from somf_MCollectible> key;
<Your Class which inherits from somf_MCollectible> value;
Environment *ev;
ev = somGetGlobalEnvironment();
key = <Your Class which inherits from somf_MCollectible>New();
ht = somf_THashTableNew();
/* Add some objects to ht */
/* Determine the value associated with key */
value = _somfRetrieve(ht, ev, key);
_somFree (ht);
_somFree (key);
```

Original Class

somf_THashTable

Related Information

None.

somfSetGrowthRate Method

Purpose

Sets the growth rate of a hash table.

Note: You cannot override this method.

Syntax

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

Parameters

receiver A pointer to an object of class somf_THashTable.

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

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

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_THashTable ht;
Environment *ev;
ev = somGetGlobalEnvironment();
ht = somf_THashTableNew();
_somfSetGrowthRate(ht, ev, 20);
_somFree (ht);
```

Original Class

somf_THashTable

Related Information

somfGetGrowthRate

somfSetHashFunction Method

Purpose

Sets a hash table's hash function to a given function. By default, this pointer is set to somf_MCollectible's somfHash method (which is usually overridden in the objects that are added to the hash table). Normally, a client program does not invoke this method.

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

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

Note: You cannot override this method.

Syntax

```
void  somfSetHashFunction(
    in somf_MCollectibleHashFn fn);
```

Parameters

receiver A pointer to an object of class somf_THashTable.

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

fn A method pointer specifying a somfHash type method.

This argument should always be set to

```
somf_MCollectibleClassData.somfHash
```

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

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_THashTable ht;
Environment *ev;
ev = somGetGlobalEnvironment();
ht = somf_THashTableNew();
somf_THashTable_somfSetHashFunction(ht, ev,
                                    somf_MCollectibleClassData.somfHash);
_somFree (ht);
```

Original Class

somf_THashTable

Related Information

somfGetHashFunction

somfSetRehashThreshold Method

Purpose

Sets the rehash threshold of a hash table.

Note: You cannot override this method.

Syntax

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

Parameters

receiver A pointer to an object of class somf_THashTable.

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

threshold The rehash threshold, expressed as the percentage of how full the hash table may become before it grows in size. For example: 80 means 80%.

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_THashTable ht;
Environment *ev;
ev = somGetGlobalEnvironment();
ht = somf_THashTableNew();
_somfSetRehashThreshold(ht, ev, 80);
_somFree (ht);
```

Original Class

somf_THashTable

Related Information

somfGetRehashThreshold

somfTHashTableInitFL Method

Purpose

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

Note: You cannot override this method.

Syntax

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

Parameters

receiver A pointer to an object of class somf_THashTable.

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

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

This argument should always be set to either

`somf_MCollectibleClassData.somfIsSame or somf_MCollectibleClassData.somfIsEqual.`

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

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

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to an initialized somf_THashTable object.

Example

```
somf_THashTable h3;
Environment *ev;
ev = somGetGlobalEnvironment();
h3 = somf_THashTableNew();
_somfTHashTableInitFL(h3, ev,
              somf_MCollectibleClassData.somfIsEqual, 23);
_somFree (h3);
```

Original Class

somf_THashTable

Related Information

somfTHashTableInitFLLL
somfTHashTableInitFLL
somfTHashTableInitH

somfTHashTableInitFLL Method

Purpose

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

Note: You cannot override this method.

Syntax

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

Parameters

receiver A pointer to an object of class somf_THashTable.

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

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

This argument should always be set to either

`somf_MCollectibleClassData.somfIsSame or somf_MCollectibleClassData.somfIsEqual.`

because SOM needs a pointer to the original declaration of the method, which resides in somf_MCollectible. The somf_THashTable object will use this pointer to access the somfIsSame Method or somfIsEqual Method that was declared and defined in the object being inserted into, or removed from, the somf_THashTable object.

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

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

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to an initialized somf_THashTable object.

Example

```
somf_THashTable h2;
Environment *ev;
ev = somGetGlobalEnvironment();
h2 = somf_THashTableNew();
_somfTHashTableInitFLL(h2, ev,
          somf_MCollectibleClassData.somfIsEqual, 23, 20);
_somFree (h2);
```

Original Class

somf_THashTable

Related Information

somfTHashTableInitFLLL
somfTHashTableInitFL
somfTHashTableInitH

somfTHashTableInitFLLL Method

Purpose

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

Note: You cannot override this method.

Syntax

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

Parameters

receiver A pointer to an object of class somf_THashTable.

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

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

This argument should always be set to either

`somf_MCollectibleClassData.somfIsSame or somf_MCollectibleClassData.somfIsEqual.`

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

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

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

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

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to an initialized somf_THashTable object.

Example

```
somf_THashTable h1;
Environment *ev;
ev = somGetGlobalEnvironment();
h1 = somf_THashTableNew();
_somfTHashTableInitFLLL(h1, ev,
      somf_MCollectibleClassData.somfIsEqual, 23, 20, 80);
_somFree (h1);
```

Original Class

somf_THashTable

Related Information

somfTHashTableInitFLL
somfTHashTableInitFL
somfTHashTableInitH

somfTHashTableInitH Method

Purpose

Initializes a new hash table, setting it equal to another specified hash table. The method also sets the new hash table equal to another specified hash table. This implies that the instance data of the new hash table will be set equal to those of the source hash table.

Note: You cannot override this method.

Syntax

```
somf_THashTable
somfTHashTableInitH(
    in somf_THashTable h);
```

Parameters

receiver A pointer to an object of class somf_THashTable.

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

h A pointer to the hash table the receiving object will be equal to.

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to an initialized somf_THashTable object.

Example

```
somf_THashTable h4;
somf_THashTable h2;
Environment *ev;
ev = somGetGlobalEnvironment();
h2 = somf_THashTableNew();
h4 = somf_THashTableNew();
_somfTHashTableInitH(h4, ev, h2);
_somFree (h2);
_somFree (h4);
```

Original Class

somf_THashTable

Related Information

somfTHashTableInitFLLL
somfTHashTableInitFLL
somfTHashTableInitFL

Chapter 13. somf_THashTableIterator Class

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

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

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

Member Name:

thashitr

Import Name:

GOSSOMUC

Base Class:

somf_TIterator Class

Metaclass:

SOMClass

Ancestor Classes:

somf_TIterator Class
SOMObject

Subclasses:

None.

Types:

None.

Attributes:

None.

New Methods:

somfTHashTableIteratorInit

Overriding Methods:

somDefaultInit
somDestruct
somfFirst
somfNext
somfRemove

somfFirst Method

Purpose

Resets the iterator and returns the first (key, value) pair of a hash table. This is true not only for the first time you use the iterator; it is also true if other operations on the hash table cause the iterator to be invalidated. In the second case, the method also revalidates the iterator.

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

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

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

Syntax

somf_MCollectible **somfFirst** ();

Parameters

receiver A pointer to an object of class somf_THashTableIterator.

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

Restrictions and Limitations

None.

Returned Values

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

Example

```
somf_THashTable ht;
Environment *ev;
somf_THashTableIterator itr;
somf_TAssoc itrobj;
somf_MCollectible objk;
somf_MCollectible objv;
ev = somGetGlobalEnvironment();
ht = somf_THashTableNew();
itr = somf_THashTableIteratorNew();
_somfTHashTableIteratorInit(itr, ev, ht);
/* Add some object to d */
/* Iterate through the THashTable */
itrobj = somf_THashTableIterator_somfFirst(itr,ev);
while (itrobj != SOMF_NIL)
{
    objk = _somfGetKey(itrobj,ev);
    objv = _somfGetValue(itrobj,ev);
   /* Do something with objk or objv */
    itrobj = _somfNext(itr,ev);
}
_somFree (ht);
_somFree (itr);
```

Original Class

somf_TIterator

Related Information

somfNext

somfNext Method

Purpose

Gets the next (key, value) pair from the hash table of a given hash-table iterator. The method returns a pointer to the next (key, value) pair, if found. Objects are retrieved in an order that reflects the "ordered-ness" of the hash table.

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

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

C cannot handle methods from different classes having the same name when they inherit the name from different parents. If any child of somf_TIterator is used with a child of somf_TPrimitiveLinkedListIterator, then the name of the method will have to be fully qualified. This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

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

Syntax

```
somf_MCollectible  somfNext ();
```

Parameters

receiver A pointer to an object of class somf_THashTableIterator.

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

Restrictions and Limitations

None.

Returned Values

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

SOMF_NIL The end of the collection has been reached.

Example

```
somf_THashTable ht;
Environment *ev;
somf_THashTableIterator itr;
somf_TAssoc itrobj;
somf_MCollectible objk;
somf_MCollectible objv;
ev = somGetGlobalEnvironment();
ht = somf_THashTableNew();
itr = somf_THashTableIteratorNew();
_somfTHashTableIteratorInit(itr, ev, ht);
/* Add some object to d */
/* Iterate through the THashTable */
itrobj = somf_THashTableIterator_somfFirst(itr,ev);
while (itrobj != SOMF_NIL)
{
    objk = _somfGetKey(itrobj,ev);
    objv = _somfGetValue(itrobj,ev);
   /* Do something with objk or objv */
    itrobj = _somfNext(itr,ev);
}
_somFree (ht);
_somFree (itr);
```

Original Class

somf_TIterator

Related Information

somfFirst

somfRemove Method

Purpose

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

The somfRemove method is the only way to remove a (key, value) object from a hash table during iteration. However, if multiple iterators are in process, all the other iterators are invalidated, just as if some other kind of change had occurred in the hash table. If the hash table has changed since the last time somfFirst was called (other than through the use of the somfRemove method of this iterator), this method will fail.

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

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

Syntax

```
void  somfRemove ();
```

Parameters

receiver A pointer to an object of class somf_THashTableIterator.

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

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_THashTable ht;
Environment *ev;
somf_THashTableIterator itr;
somf_MCollectible itrobj;
ev = somGetGlobalEnvironment();
ht = somf_THashTableNew();
itr = somf_THashTableIteratorNew();
_somfTHashTableIteratorInit(itr, ev, ht);
/* Add some objects to ht */
/* Use the Iterator's Remove to remove the first object */
itrobj = somf_THashTableIterator_somfFirst(itr,ev);
somf_THashTableIterator_somfRemove(itr,ev);
_somFree (ht);
_somFree (itr);
```

Original Class

somf_TIterator

Related Information

None.

somfTHashTableIteratorInit Method

Purpose

Initializes a somf_THashTableIterator iterator, given its corresponding hash table.

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

Note: You cannot override this method.

```
somf_THashTableIterator
somfTHashTableIteratorInit(
    in somf_THashTable h);
```

Parameters

receiver A pointer to an object of class somf_THashTableIterator.

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

h A pointer to the hash table the receiving object will iterate over.

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to an initialized somf_THashTableIterator object.

Example

```
somf_THashTable ht;
Environment *ev;
somf_THashTableIterator itr;
ev = somGetGlobalEnvironment();
ht = somf_THashTableNew();
itr = somf_THashTableIteratorNew();
_somfTHashTableIteratorInit(itr, ev, ht);
_somFree (ht);
_somFree (itr);
```

Original Class

somf_THashTableIterator

Related Information

None.

Chapter 14. somf_TIterator Class

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

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

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

Member Name:

titeratr

Import Name:

GOSSOMUC

Base Class:

SOMObject

Metaclass:

SOMClass

Ancestor Classes:

SOMObject

Subclasses:

None.

Types:

None.

Attributes:

None.

New Methods:

somfNext
somfFirst
somfRemove

Overriding Methods:

None.

somfFirst Method

Purpose

Resets the iterator and returns the first object of a collection. This is true not only for the first time you use the iterator; it is also true if other operations on the collection cause the iterator to be invalidated. In the second case, the method also revalidates the iterator.

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

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

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

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

Syntax

```
somf_MCollectible  somfFirst ();
```

Parameters

receiver A pointer to an object of class somf_TIterator.

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

Restrictions and Limitations

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.

Returned Values

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

Example

For examples of how this method looks when it is invoked, see somf_TSetIterator Class or somf_TDictionaryIterator Class, or any of the other classes that inherit from somf_TIterator.

Original Class

somf_TIterator

Related Information

somfNext

somfNext Method

Purpose

Gets the next object in a collection. Objects are retrieved in an order that reflects the "ordered-ness" of the collection (or the lack of ordering on the collection objects).

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

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

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

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

Syntax

somf_MCollectible **somfNext** ();

Parameters

receiver A pointer to an object of class somf_TIterator.

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

Restrictions and Limitations

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.

Returned Values

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

SOMF_NIL The end of the collection has been reached.

Example

For examples of how this method looks when it is invoked, see somf_TSetIterator Class or somf_TDictionaryIterator Class, or any of the other classes that inherit from somf_TIterator.

Original Class

somf_TIterator

Related Information

somfFirst

somfRemove Method

Purpose

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

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

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

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

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

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

Syntax

```
void  somfRemove ();
```

Parameters

receiver A pointer to an object of class somf_TIterator.

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

Restrictions and Limitations

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.

Returned Values

None.

Example

For examples of how this method looks when it is invoked, see somf_TSetIterator Class or yIterator Class, or any of the other classes that inherit from somf_TIterator.

Original Class

somf_TIterator

Related Information

None.

Chapter 15. somf_TPrimitiveLinkedList Class

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

Objects that are inserted into a collection object of the somf_TPrimitiveLinkedList class

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

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

Member Name:

tpll

Import Name:

GOSSOMUC

Base Class:

SOMObject

Metaclass:

SOMClass

Ancestor Classes:

SOMObject

Subclasses:

None.

Types:

None.

Attributes:

None.

New Methods:

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

Overriding Methods:

somDefaultInit
somDestruct

somfAddAfter Method

Purpose

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

Syntax

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

Parameters

receiver A pointer to an object of class somf_TPrimitiveLinkedList.

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

existing A pointer to the somf_MLinkable object that obj will be added in front of.

obj A pointer to the somf_MLinkable object that will be added.

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_TPrimitiveLinkedList l;
<Your Class which inherits from MLinkable> obj;
<Your Class which inherits from MLinkable> obj2;
Environment *ev;
ev = somGetGlobalEnvironment();
l = somf_TPrimitiveLinkedListNew();
obj = <Your Class which inherits from MLinkable>New();
obj2 = <Your Class which inherits from MLinkable>New();
/* Add obj2 to l after obj */
_somfAddFirst(l, ev, obj);
_somfAddAfter(l, ev, obj, obj2);
_somFree (l);
_somFree (obj);
_somFree (obj2);
```

Original Class

somf_TPrimitiveLinkedList

Related Information

somfAddBefore
somfAddFirst
somfAddLast

somfAddBefore Method

Purpose

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

Syntax

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

Parameters

receiver A pointer to an object of class somf_TPrimitiveLinkedList.

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

existing pointer to the somf_MLinkable object that obj will be added in front of.

obj A pointer to the somf_MLinkable object that will be added.

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_TPrimitiveLinkedList l;
<Your Class which inherits from MLinkable> obj;
<Your Class which inherits from MLinkable> obj2;
Environment *ev;
ev = somGetGlobalEnvironment();
l = somf_TPrimitiveLinkedListNew();
obj = <Your Class which inherits from MLinkable>New();
obj2 = <Your Class which inherits from MLinkable>New();
/* Add obj2 to l before obj */
_somfAddFirst(l, ev, obj);
_somfAddBefore(l, ev, obj, obj2);
_somFree (l);
_somFree (obj);
_somFree (obj2);
```

Original Class

somf_TPrimitiveLinkedList

Related Information

somfAddAfter
somfAddFirst
somfAddLast

somfAddFirst Method

Purpose

Adds an object as the first object in a list.

Syntax

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

Parameters

receiver A pointer to an object of class somf_TPrimitiveLinkedList.

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

obj A pointer to the somf_MLinkable that will be added.

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_TPrimitiveLinkedList l;
<Your Class which inherits from MLinkable> obj;
Environment *ev;
ev = somGetGlobalEnvironment();
l = somf_TPrimitiveLinkedListNew();
obj = <Your Class which inherits from MLinkable>New();
/* Add obj to the front of l */
_somfAddFirst(l, ev, obj);
_somFree (l);
_somFree (obj);
```

Original Class

somf_TPrimitiveLinkedList

Related Information

somfAddAfter
somfAddBefore
somfAddLast

somfAddLast Method

Purpose

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

Syntax

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

Parameters

receiver A pointer to an object of class somf_TPrimitiveLinkedList.

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

obj A pointer to the somf_MLinkable that will be added.

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_TPrimitiveLinkedList l;
<Your Class which inherits from MLinkable> obj;
Environment *ev;
ev = somGetGlobalEnvironment();
l = somf_TPrimitiveLinkedListNew();
obj = <Your Class which inherits from MLinkable>New();
/* Add obj to the end of l */
_somfAddLast(l, ev, obj);
_somFree (l);
_somFree (obj);
```

Original Class

somf_TPrimitiveLinkedList

Related Information

somfAddAfter
somfAddBefore
somfAddFirst

somfAfter Method

Purpose

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

Syntax

```
somf_MLinkable  somfAfter(
    in somf_MLinkable existingobj);
```

Parameters

receiver A pointer to an object of class somf_TPrimitiveLinkedList.

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

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

Restrictions and Limitations

None.

Returned Values

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

SOMF_NIL Nothing is after *existingobj*.

Example

```
somf_TPrimitiveLinkedList l;
<Your Class which inherits from MLinkable> obj;
somf_MLinkable obj2;
Environment *ev;
ev = somGetGlobalEnvironment();
l = somf_TPrimitiveLinkedListNew();
obj = <Your Class which inherits from MLinkable>New();
/* Add a lot of objects to l */
/* Determine the object in l after obj */
obj2 = _somfAfter(l, ev, obj);
_somFree (l);
_somFree (obj);
```

Original Class

somf_TPrimitiveLinkedList

Related Information

somfBefore

somfBefore Method

Purpose

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

Syntax

```
somf_MLinkable  somfBefore(
    in somf_MLinkable existingobj);
```

Parameters

receiver A pointer to an object of class somf_TPrimitiveLinkedList.

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

existingobj A pointer to the somf_MLinkable object that comes after the returned *obj*.

Restrictions and Limitations

None.

Returned Values

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

SOMF_NIL Nothing is before the *existingobj*.

Example

```
somf_TPrimitiveLinkedList l;
<Your Class which inherits from MLinkable> obj;
somf_MLinkable obj2;
Environment *ev;
ev = somGetGlobalEnvironment();
l = somf_TPrimitiveLinkedListNew();
obj = <Your Class which inherits from MLinkable>New();
/* Add a lot of objects to l */
/* Determine the object in l before obj */
obj2 = _somfBefore(l, ev, obj);
_somFree (l);
_somFree (obj);
```

Original Class

somf_TPrimitiveLinkedList

Related Information

somfAfter

somfCount Method

Purpose

Gets the number of objects in a given list.

Syntax

```
unsigned long  somfCount ();
```

Parameters

receiver A pointer to an object of class somf_TPrimitiveLinkedList.

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

Restrictions and Limitations

None.

Returned Values

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

Example

```
somf_TPrimitiveLinkedList l;
Environment *ev;
ev = somGetGlobalEnvironment();
l = somf_TPrimitiveLinkedListNew();
/* Add some objects to l */
/* Print the number of objects in ht */
somPrintf("\n Count of l= d\n",
          somf_TPrimitiveLinkedList_somfCount(l,ev));
_somFree (l);
```

Original Class

somf_TPrimitiveLinkedList

Related Information

None.

somfFirst Method

Purpose

Gets the first object in a given list.

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

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

Syntax

somf_MLinkable **somfFirst** ();

Parameters

receiver A pointer to an object of class somf_TPrimitiveLinkedList.

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

Restrictions and Limitations

None.

Returned Values

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

SOMF_NIL Nothing is in the list.

Example

```
somf_TPrimitiveLinkedList l;
somf_MLinkable obj;
Environment *ev;
ev = somGetGlobalEnvironment();
l = somf_TPrimitiveLinkedListNew();
/* Add a lot of objects to l */
/* Determine the first object in l */
obj = somf_TPrimitiveLinkedList_somfFirst(l,ev);
_somFree (l);
```

Original Class

somf_TPrimitiveLinkedList

Related Information

somfLast

somfLast Method

Purpose

Gets the last object in a given list.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. somfLast is a method name declared in multiple parents (for example: somf_TSequenceIterator, somf_TSequence). You will probably have to fully qualify the name of the method. This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

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

Syntax

somf_MLinkable **somfLast** ();

Parameters

receiver A pointer to an object of class somf_TPrimitiveLinkedList.

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

Restrictions and Limitations

None.

Returned Values

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

SOMF_NIL Nothing is in the list.

Example

```
somf_TPrimitiveLinkedList l;
somf_MLinkable obj;
Environment *ev;
ev = somGetGlobalEnvironment();
l = somf_TPrimitiveLinkedListNew();
/* Add a lot of objects to l */
/* Determine the last object in l */
obj = somf_TPrimitiveLinkedList_somfLast(l,ev);
_somFree (l);
```

Original Class

somf_TPrimitiveLinkedList

Related Information

somfFirst

somfRemove Method

Purpose

Removes a somf_MLinkable object from a given list.

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

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

Syntax

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

Parameters

receiver A pointer to an object of class somf_TPrimitiveLinkedList.

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

aLink A pointer to the somf_MLinkable object to be removed.

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_TPrimitiveLinkedList l;
<Your Class which inherits from MLinkable> obj;
Environment *ev;
ev = somGetGlobalEnvironment();
l = somf_TPrimitiveLinkedListNew();
obj = <Your Class which inherits from MLinkable>New();
/* Add a lot of objects to l */
/* Remove obj from l */
somf_TPrimitiveLinkedList_somfRemove(l, ev, obj);
_somFree (l);
_somFree (obj);
```

Original Class

somf_TPrimitiveLinkedList

Related Information

somfRemoveAll
somfRemoveFirst
somfRemoveLast

somfRemoveAll Method

Purpose

Removes all of the objects from a given list.

Syntax

```
void  somfRemoveAll ();
```

Parameters

receiver A pointer to an object of class somf_TPrimitiveLinkedList.

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

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_TPrimitiveLinkedList l;
<Your Class which inherits from MLinkable> obj;
Environment *ev;
ev = somGetGlobalEnvironment();
l = somf_TPrimitiveLinkedListNew();
obj = <Your Class which inherits from MLinkable>New();
/* Add a lot of objects to l */
/* Remove all of the objects from l */
somf_TPrimitiveLinkedList_somfRemoveAll(l,ev);
_somFree (l);
_somFree (obj);
```

Original Class

somf_TPrimitiveLinkedList

Related Information

somfRemove
somfRemoveFirst
somfRemoveLast

somfRemoveFirst Method

Purpose

Removes the first object from a given list.

Syntax

```
somf_MLinkable  somfRemoveFirst ();
```

Parameters

receiver A pointer to an object of class somf_TPrimitiveLinkedList.

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

Restrictions and Limitations

None.

Returned Values

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

SOMF_NIL Nothing is in the list.

Example

```
somf_TPrimitiveLinkedList l;
Environment *ev;
ev = somGetGlobalEnvironment();
l = somf_TPrimitiveLinkedListNew();
/* Add some objects to l */
/* Remove the first object */
if (_somfRemoveFirst(l,ev) == SOMF_NIL)
    somPrintf(" The list is empty\n");
_somFree (l);
```

Original Class

somf_TPrimitiveLinkedList

Related Information

somfRemove
somfRemoveAll
somfRemoveLast

somfRemoveLast Method

Purpose

Removes the last object from a given list.

Syntax

```
somf_MLinkable  somfRemoveLast ();
```

Parameters

receiver A pointer to an object of class somf_TPrimitiveLinkedList.

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

Restrictions and Limitations

None.

Returned Values

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

SOMF_NIL Nothing is in the list.

Example

```
somf_TPrimitiveLinkedList l;
Environment *ev;
ev = somGetGlobalEnvironment();
l = somf_TPrimitiveLinkedListNew();
/* Add some objects to l */
/* Remove the last object */
if (_somfRemoveLast(l,ev) == SOMF_NIL)
    somPrintf(" The list is empty\n");
_somFree (l);
```

Original Class

somf_TPrimitiveLinkedList

Related Information

somfRemove
somfRemoveAll
somfRemoveFirst

Chapter 16. somf_TPrimitiveLinkedListIterator Class

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

Member Name:

tpllitr

Import Name:

GOSSOMUC

Base Class:

SOMObject

Metaclass:

SOMClass

Ancestor Classes:

SOMObject

Subclasses:

None.

Types:

None.

Attributes:

None.

New Methods:

somfFirst
somfNext
somfLast
somfPrevious
somfTPrimitiveLinkedListIteratorInit

Overriding Methods:

somDestruct

somfFirst Method

Purpose

Resets the iterator and returns the first element of a given list. The somf_TPrimitiveLinkedListIterator class does not inherit from somf_TIterator Class. This method may look like the somf_TIterator method, but there is no connection.

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

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

Syntax

```
somf_MLinkable  somfFirst ();
```

Parameters

receiver A pointer to an object of class somf_TPrimitiveLinkedListIterator.

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

Restrictions and Limitations

None.

Returned Values

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

SOMF_NIL Nothing is in the list.

Example

```
somf_TPrimitiveLinkedList l;
somf_MLinkable obj;
somf_TPrimitiveLinkedListIterator itr;
Environment *ev;
ev = somGetGlobalEnvironment();
l = somf_TPrimitiveLinkedListNew();
itr = somf_TPrimitiveLinkedListIteratorNew();
_somfTPrimitiveLinkedListIteratorInit(itr, ev, l);
/* Add a lot of objects to l */
/* Iterate through l */
obj = somf_TPrimitiveLinkedListIterator_somfFirst(itr,ev);
while (obj != SOMF_NIL)
{
    /* do something with obj */
    obj = _somfNext(itr,ev);
}
_somFree (l);
_somFree (itr);
```

Original Class

somf_TPrimitiveLinkedListIterator

Related Information

somfNext

somfLast Method

Purpose

Retrieves the last object from a given list.

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

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

Syntax

somf_MLinkable **somfLast** ();

Parameters

receiver A pointer to an object of class somf_TPrimitiveLinkedListIterator.

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

Restrictions and Limitations

None.

Returned Values

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

SOMF_NIL Nothing is in the list.

Example

```
somf_TPrimitiveLinkedList l;
somf_MLinkable obj;
somf_TPrimitiveLinkedListIterator itr;
Environment *ev;
ev = somGetGlobalEnvironment();
l = somf_TPrimitiveLinkedListNew();
itr = somf_TPrimitiveLinkedListIteratorNew();
_somfTPrimitiveLinkedListIteratorInit(itr, ev, l);
/* Add a lot of objects to l */
/* Find the last object in l */
obj = somf_TPrimitiveLinkedList_somfLast(l,ev);
_somFree (l);
_somFree (itr);
```

Original Class

somf_TPrimitiveLinkedListIterator

Related Information

somfPrevious

somfNext Method

Purpose

Gets the next object in a list. The somf_TPrimitiveLinkedListIterator class does not inherit from somf_TIterator Class.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. If any child of somf_TPrimitiveLinkedListIterator is used with somf_TIterator, then the name of the method will have to be fully qualified. This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

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

Syntax

somf_MLinkable **somfNext** ();

Parameters

receiver A pointer to an object of class somf_TPrimitiveLinkedListIterator.

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

Restrictions and Limitations

None.

Returned Values

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

SOMF_NIL The end of the list has been reached.

Example

```
somf_TPrimitiveLinkedList l;
somf_MLinkable obj;
somf_TPrimitiveLinkedListIterator itr;
Environment *ev;
ev = somGetGlobalEnvironment();
l = somf_TPrimitiveLinkedListNew();
itr = somf_TPrimitiveLinkedListIteratorNew();
_somfTPrimitiveLinkedListIteratorInit(itr, ev, l);
/* Add a lot of objects to l */
/* Iterate through l */
obj = somf_TPrimitiveLinkedListIterator_somfFirst(itr,ev);
while (obj != SOMF_NIL)
{
    /* do something with obj */
    obj = _somfNext(itr,ev);
}
_somFree (l);
_somFree (itr);
```

Original Class

somf_TPrimitiveLinkedListIterator

Related Information

somfFirst

somfPrevious Method

Purpose

Gets the previous object from a given list.

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

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

Syntax

somf_MLinkable **somfPrevious** ();

Parameters

receiver A pointer to an object of class somf_TPrimitiveLinkedListIterator.

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

Restrictions and Limitations

None.

Returned Values

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

SOMF_NIL The beginning of the list has been reached.

Example

```
somf_TPrimitiveLinkedList l;
somf_MLinkable obj;
somf_TPrimitiveLinkedListIterator itr;
Environment *ev;
ev = somGetGlobalEnvironment();
l = somf_TPrimitiveLinkedListNew();
itr = somf_TPrimitiveLinkedListIteratorNew();
_somfTPrimitiveLinkedListIteratorInit(itr, ev, l);
/* Add a lot of objects to l */
/* Find the next to the last object in l */
somf_TPrimitiveLinkedList_somfLast(l,ev);
obj = _somfPrevious(next,ev);
_somFree (l);
_somFree (itr);
```

Original Class

somf_TPrimitiveLinkedListIterator

Related Information

somfLast

somfTPrimitiveLinkedListIteratorInit Method

Purpose

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

Note: You cannot override this method.

Syntax

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

Parameters

receive A pointer to an object of class somf_TPrimitiveLinkedListIterator.

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

list A pointer to the primitive linked list object that the receiving object will iterate over.

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to an initialized somf_TPrimitiveLinkedListIterator iterator.

Example

```
somf_TPrimitiveLinkedList l;
Environment *ev;
somf_TPrimitiveLinkedListIterator itr;
ev = somGetGlobalEnvironment();
l = somf_TPrimitiveLinkedListNew();
itr = somf_TPrimitiveLinkedListIteratorNew();
_somfTPrimitiveLinkedListIteratorInit(itr, ev, l);
_somFree (l);
_somFree (itr);
```

Original Class

somf_TPrimitiveLinkedListIterator

Related Information

None.

Chapter 17. somf_TPriorityQueue Class

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

In *Algorithms in C++* Robert Sedgewick describes a *priority queue* as follows:

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

Note: The somf_TPriorityQueue class uses the somfIsEqual: epk. 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 somIsSame method.

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

Member Name:

tpq

Import Name:

GOSSOMUC

Base Class:

somf_TCollection Class

Metaclass:

SOMClass

Ancestor Classes:

somf_TCollection
somf_MCollectible
SOMObject

Subclasses:

None.

Types:

None.

Attributes:

None.

New Methods:

somfInsert
somfPeek
somfPop
somfReplace
somfSetEqualityComparisonFunction
somfGetEqualityComparisonFunction
somfAssign
somfTPriorityQueueInitF
somfTPriorityQueueInitP

Overriding Methods:

somDefaultInit
somDestruct
somfAdd
somfRemove
somfRemoveAll
somfDeleteAll
somfCount
somfMember
somfCreateIterator

somfAdd Method

Purpose

Adds a given *obj* to a priority queue.

Syntax

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

Parameters

receiver A pointer to an object of class somf_TPriorityQueue.

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

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

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to the somf_MCollectible object added.

Example

```
somf_TPriorityQueue pq;
<Your Class which inherits from somf_MOrderableCollectible> obj;
Environment *ev;
ev = somGetGlobalEnvironment();
pq = somf_TPriorityQueueNew();
obj = <Your Class which inherits from
       somf_MOrderableCollectible>New();
/* Add obj to pq */
_somfAdd(pq,ev,obj);
_somFree (pq);
_somFree (obj);
```

Original Class

somf_TCollection (overridden here)

Related Information

somfInsert

somfAssign Method

Purpose

Assigns a priority-queue receiving object as being equal to a given source priority queue. That is, the method sets/resets the instance variables of the receiver to the values of the source. This operation is logically equivalent to using the equal (=) operator.

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

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

Syntax

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

Parameters

receiver A pointer to an object of class somf_TPriorityQueue.

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

source A pointer to the somf_TPriorityQueue object the receiving object will be equal to.

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_TPriorityQueue pq1;
somf_TPriorityQueue pq2;
Environment *ev;
ev = somGetGlobalEnvironment();
pq1 = somf_TPriorityQueueNew();
pq2 = somf_TPriorityQueueNew();
/* Add some objects to pq1 */
/* Assign pq2 = pq1 */
somf_TPriorityQueue_somfAssign(pq2,ev,pq1);
_somFree (pq1);
_somFree (pq2);
```

Original Class

somf_TPriorityQueue

Related Information

None.

somfCount Method

Purpose

Gets the number of objects in a given priority queue.

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

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

Syntax

long **somfCount** ();

Parameters

receiver A pointer to an object of class somf_TPriorityQueue.

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

Restrictions and Limitations

None.

Returned Values

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

Example

```
somf_TPriorityQueue pq;
Environment *ev;
ev = somGetGlobalEnvironment();
pq = somf_TPriorityQueueNew();
/* Add some objects to pq */
/* Count the number of objects in pq */
somPrintf("\n Count of pq= %d\n", _somfCount(pq,ev));
_somFree (pq);
```

Original Class

somf_TCollection (overridden here)

Related Information

None.

somfCreateIterator Method

Purpose

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

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

Syntax

somf_TIterator **somfCreateIterator** ();

Parameters

receiver A pointer to an object of class somf_TPriorityQueue.

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

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to the new iterator.

Example

```
somf_TPriorityQueue pq;
Environment *ev;
somf_TPriorityQueueIterator itr;
ev = somGetGlobalEnvironment();
pq = somf_TPriorityQueueNew();
itr = (somf_TPriorityQueueIterator*)
      _somfCreateIterator(pq,ev);
_somFree (pq);
_somFree (itr);
```

Original Class

somf_TCollection (overridden here)

Related Information

None.

somfDeleteAll Method

Purpose

Removes all of the objects from a priority-queue receiving object and deallocates the storage that these objects might have owned. (That is, the destructor function is called for each object in the collection.) The method also deallocates the storage that these objects might have owned (that is, the destructor function is called for each object in the collection).

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

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

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

Syntax

```
void  somfDeleteAll ();
```

Parameters

receiver A pointer to an object of class somf_TPriorityQueue.

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

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_TPriorityQueue pq;
Environment *ev;
ev = somGetGlobalEnvironment();
pq = somf_TPriorityQueueNew();
/* Add objects to pq */
/* Remove all the objects from pq AND DELETE THEM */
_somfDeleteAll(pq,ev);
_somFree (pq);
```

Original Class

somf_TCollection (overridden here)

Related Information

None.

somfGetEqualityComparisonFunction Method

Purpose

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

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

Syntax

somf_MCollectibleCompareFn
somfGetEqualityComparisonFunction ();

Parameters

receiver A pointer to an object of class somf_TPriorityQueue.

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

Restrictions and Limitations

None.

Returned Values

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

Example

```
somf_TPriorityQueue pq;
Environment *ev;
ev = somGetGlobalEnvironment();
pq = somf_TPriorityQueueNew();
/* Add some objects to pq */
if (_somfGetEqualityComparisonFunction(pq,ev) !=
    somf_MCollectibleClassData.somfIsEqual)
{
    somPrintf("\n What Compare Function are we using?\n");
}
_somFree (pq);
```

Original Class

somf_TPriorityQueue

Related Information

somfSetEqualityComparisonFunction

somfInsert Method

Purpose

Inserts an object *obj* into the priority queue. This method is just like the somfAdd method, except that it does not return a pointer to the somf_MCollectible object added.

Syntax

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

Parameters

receiver A pointer to an object of class somf_TPriorityQueue.

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

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

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_TPriorityQueue pq;
<Your Class which inherits from somf_MOrderableCollectible> obj;
Environment *ev;
ev = somGetGlobalEnvironment();
pq = somf_TPriorityQueueNew();
obj = <Your Class which inherits from
       somf_MOrderableCollectible>New();
/* Add obj to pq */
_somfInsert(pq,ev,obj);
_somFree (pq);
_somFree (obj);
```

Original Class

somf_TPriorityQueue

Related Information

somfAdd

somfMember Method

Purpose

Gets an object from a given priority queue.

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

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

Syntax

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

Parameters

receiver A pointer to an object of class somf_TPriorityQueue.

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

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

Restrictions and Limitations

None.

Returned Values

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

SOMF_NIL The object was not found.

Example

```
somf_TPriorityQueue pq;
<Your Class which inherits from somf_MOrderableCollectible> obj;
Environment *ev;
ev = somGetGlobalEnvironment();
pq = somf_TPriorityQueueNew();
obj = <Your Class which inherits from
       somf_MOrderableCollectible>New();
/* Add some objects to pq */
/* See if obj is in pq */
if (somf_TPriorityQueue_somfMember(pq, ev, obj) == SOMF_NIL)
    somPrintf("\n obj is NOT in d\n");
else
    somPrintf("\n obj IS in d\n");
_somFree (pq);
```

Original Class

somf_TCollection (overridden here)

Related Information

None.

somfPeek Method

Purpose

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

Syntax

```
somf_MOrderableCollectible
somfPeek ();
```

Parameters

receiver A pointer to an object of class somf_TPriorityQueue.

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

Restrictions and Limitations

None.

Returned Values

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

SOMF_NIL No object remains in the priority queue.

Example

```
somf_TPriorityQueue pq;
somf_MOrderableCollectible obj;
Environment *ev;
ev = somGetGlobalEnvironment();
pq = somf_TPriorityQueueNew();
/* Add some objects to pq */
/* Look at the highest priority object */
if ((obj = (_somfPeek(pq,ev))) == SOMF_NIL)
    somPrintf(" Nothing is in pq\n");
_somFree (pq);
_somFree (obj);
```

Original Class

somf_TPriorityQueue

Related Information

somfPop

somfPop Method

Purpose

Gets the object with the "highest" priority from a given priority queue.

Syntax

```
somf_MOrderableCollectible
somfPop ();
```

Parameters

receiver A pointer to an object of class somf_TPriorityQueue.

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

Restrictions and Limitations

None.

Returned Values

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

SOMF_NIL No object remains in the priority queue.

Example

```
somf_TPriorityQueue pq;
somf_MOrderableCollectible obj;
Environment *ev;
ev = somGetGlobalEnvironment();
pq = somf_TPriorityQueueNew();
/* Add some objects to pq */
/* Get the highest priority object */
if ((obj = (_somfPop(pq,ev))) == SOMF_NIL)
    somPrintf(" Nothing is in pq\n");
_somFree (pq);
_somFree (obj);
```

Original Class

somf_TPriorityQueue

Related Information

somfPeek
somfReplace

somfRemove Method

Purpose

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

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

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

Syntax

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

Parameters

receiver A pointer to an object of class somf_TPriorityQueue.

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

obj A pointer to the somf_MCollectible object to be removed from the priority queue.

Restrictions and Limitations

None.

Returned Values

somf_MCollectible
A pointer to the object that was actually removed.

SOMF_NIL The specified object was not found.

Example

```
somf_TPriorityQueue pq;
<Your Class which inherits from somf_MOrderableCollectible> obj;
Environment *ev;
ev = somGetGlobalEnvironment();
pq = somf_TPriorityQueueNew();
obj = <Your Class that inherits from
       somf_MOrderableCollectible>New();
/* Add objects to pq */
/* Remove obj from pq */
if (somf_TPriorityQueue_somfRemove(pq,ev,obj) == SOMF_NIL)
    somPrintf(" obj was not in pq\n");
_somFree (pq);
_somFree (obj);
```

Original Class

somf_TCollection (overridden here)

Related Information

somfRemoveAll

somfRemoveAll Method

Purpose

Removes all of the objects from a given priority queue.

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

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

Syntax

void **somfRemoveAll** ();

Parameters

receiver A pointer to an object of class somf_TPriorityQueue.

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

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_TPriorityQueue pq;
Environment *ev;
ev = somGetGlobalEnvironment();
pq = somf_TPriorityQueueNew();
/* Add objects to pq */
/* Remove all the objects from pq */
_somfRemoveAll(pq,ev);
_somFree (pq);
```

Original Class

somf_TCollection (overridden here)

Related Information

somfRemove

somfReplace Method

Purpose

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

Syntax

```
somf_MOrderableCollectible
somfReplace(
    in somf_MOrderableCollectible
    obj);
```

Parameters

receiver A pointer to an object of class somf_TPriorityQueue.

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

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

Restrictions and Limitations

None.

Returned Values

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

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

Example

```
somf_TPriorityQueue pq;
<Your Class which inherits from somf_MOrderableCollectible> obj;
Environment *ev;
ev = somGetGlobalEnvironment();
pq = somf_TPriorityQueueNew();
obj = <Your Class which inherits from
       somf_MOrderableCollectible>New();
/* Add objects to pq */
if ((_somfReplace(pq,ev,obj)) == SOMF_NIL)
    somPrintf(" pq was empty\n");
_somFree (pq);
_somFree (obj);
```

Original Class

somf_TPriorityQueue

Related Information

somfPop
somfInsert

somfSetEqualityComparisonFunction Method

Purpose

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

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

Syntax

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

Parameters

receiver A pointer to an object of class somf_TPriorityQueue.

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

testfn A method pointer specifying either a somfIsEqual Method 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 Class. The somf_TPriorityQueue object will use this pointer to access the somfIsSame or somfIsEqual method that was declared and defined in the object being inserted into, or removed from, the somf_TPriorityQueue object.

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_TPriorityQueue pq;
Environment *ev;
ev = somGetGlobalEnvironment();
pq = somf_TPriorityQueueNew();
/* Add some objects to pq */
_somfSetEqualityComparisonFunction(pq,ev,
                  somf_MCollectibleClassData.somfIsEqual);
_somFree (pq);
```

Original Class

somf_TPriorityQueue

Related Information

somfGetEqualityComparisonFunction

somfTPriorityQueueInitF Method

Purpose

Initializes a new priority queue, given a comparison test method. This is the only way to set the comparison function used to determine priority for instances of the class. If this method is not used, somfIsLessThan is used.

Note: You cannot override this method.

Syntax

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

Parameters

receiver A pointer to an object of class somf_TPriorityQueue.

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

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

This should always be set to either

```
somf_MOrderableCollectibleClassData.
                                    somfIsLessThan or
somf_MOrderableCollectibleClassData.
                                    somfIsGreaterThan
```

because SOM needs a pointer to the original declaration of the method, which resides in somf_MOrderableCollectible Class. The somf_TPriorityQueue object will use this pointer to access the somfIsLessThan or somfIsGreaterThan method that was declared and defined in the inserted or removed object.

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to an initialized somf_TPriorityQueue object.

Example

```
somf_TPriorityQueue pq1;
Environment *ev;
ev = somGetGlobalEnvironment();
pq1 = somf_TPriorityQueueNew();
_somfTPriorityQueueInitF(pq1,ev,
      somf_MOrderableCollectibleClassData.somfIsLessThan);
_somFree (pq1);
```

Original Class

somf_TPriorityQueue

Related Information

somfTPriorityQueueInitP

somfTPriorityQueueInitP Method

Purpose

Initializes a new priority queue, setting it equal to another specified priority queue. The method also sets the new priority queue equal to another specified priority queue. This implies that the instance data of the new priority queue will be set equal to those of the source priority queue.

Note: You cannot override this method.

Syntax

```
somf_TPriorityQueue
somfTPriorityQueueInitP(
    in somf_TPriorityQueue q);
```

Parameters

receiver A pointer to an object of class somf_TPriorityQueue.

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

q A pointer to the existing instance of somf_TPriorityQueue to which the new priority queue will be set equal.

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to an initialized somf_TPriorityQueue object.

Example

```
somf_TPriorityQueue pq1;
somf_TPriorityQueue pq2;
Environment *ev;
ev = somGetGlobalEnvironment();
pq1 = somf_TPriorityQueueNew();
pq2 = somf_TPriorityQueueNew();
_somfTPriorityQueueInitP(pq2,ev,pq1);
_somFree (pq1);
_somFree (pq2);
```

Original Class

somf_TPriorityQueue

Related Information

somfTPriorityQueueInitF

Chapter 18. somf_TPriorityQueueIterator Class

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

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

Member Name:

tpqitr

Import Name:

GOSSOMUC

Base Class:

somf_TIterator

Metaclass:

SOMClass

Ancestor Classes:

somf_TIterator
SOMClass

Subclasses:

None.

Types:

None.

Attributes:

None.

New Methods:

somfTPriorityQueueIteratorInit

Overriding Methods:

somfNext
somfFirst
somfRemove

somfFirst Method

Purpose

Resets the iterator and returns the first object in a priority queue. The method returns the first object of the priority queue that corresponds to the specified iterator. This method resets the iterator to the beginning even if other operations on the collection cause the iterator to be invalidated.

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

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

Syntax

somf_MCollectible **somfFirst** ();

Parameters

receiver A pointer to an object of class somf_TPriorityQueueIterator.

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

Restrictions and Limitations

None.

Returned Values

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

Example

```
somf_TPriorityQueue pq;
Environment *ev;
somf_TPriorityQueueIterator itr;
somf_MOrderableCollectible itrobj;
ev = somGetGlobalEnvironment();
pq = somf_TPriorityQueueNew();
itr = somf_TPriorityQueueIteratorNew();
_somfTPriorityQueueIteratorInit(itr, ev, pq);
/* Add some object to pq */
/* Iterate through the TPriorityQueue */
itrobj = _somfFirst(itr,ev);
while (itrobj != SOMF_NIL)
{
    /* do something with itrobj */
    itrobj = _somfNext(itr,ev);
}
_somFree (pq);
_somFree (itr);
```

Original Class

somf_TIterator (overridden here)

Related Information

somfNext

somfNext Method

Purpose

Gets the next object in a priority queue.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. If any child of somf_TIterator is used with a child of somf_TPrimitiveLinkedListIterator, then the name of the method will have to be fully qualified. This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

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

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

Syntax

somf_MCollectible **somfNext** ();

Parameters

receiver A pointer to an object of class somf_TPriorityQueueIterator.

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

Restrictions and Limitations

None.

Returned Values

somf_MCollectible
A pointer to the next somf_MCollectible in the queue.

SOMF_NIL The end of the collection has been reached.

Example

```
somf_TPriorityQueue pq;
Environment *ev;
somf_TPriorityQueueIterator itr;
somf_MOrderableCollectible itrobj;
ev = somGetGlobalEnvironment();
pq = somf_TPriorityQueueNew();
itr = somf_TPriorityQueueIteratorNew();
_somfTPriorityQueueIteratorInit(itr, ev, pq);
/* Add some object to pq */
/* Iterate through the TPriorityQueue */
itrobj = _somfFirst(itr,ev);
while (itrobj != SOMF_NIL)
{
    /* do something with itrobj */
    itrobj = _somfNext(itr,ev);
}
_somFree (pq);
_somFree (itr);
```

Original Class

somf_TIterator (overridden here)

Related Information

somfFirst

somfRemove Method

Purpose

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

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

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

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

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

Syntax

```
void  somfRemove ();
```

Parameters

receiver A pointer to an object of class somf_TPriorityQueueIterator.

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

Restrictions and Limitations

None.

Example

```
somf_TPriorityQueue pq;
Environment *ev;
somf_TPriorityQueueIterator itr;
ev = somGetGlobalEnvironment();
pq = somf_TPriorityQueueNew();
itr = somf_TPriorityQueueIteratorNew();
_somfTPriorityQueueIteratorInit(itr, ev, pq);
/* Add some object to pq */
/* Remove the first object in pq */
_somfFirst(itr,ev);
somf_TPriorityQueueIterator_somfRemove(itr,ev);
_somFree (pq);
_somFree (itr);
```

Original Class

somf_TIterator

Related Information

None.

somfTPriorityQueueIteratorInit Method

Purpose

Initializes a new priority queue iterator, given the priority queue over which it will iterate. This is one of two ways to initialize a somf_TPriorityQueueIterator iterator to point to an instance of a somf_TPriorityQueue priority queue collection. The other is to use the somf_TPriorityQueue class's somfCreateIterator Method.

Note: You cannot override this method.

Syntax

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

Parameters

receiver A pointer to an object of class somf_TPriorityQueueIterator.

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

h A pointer to the somf_TPriorityQueue object over which the receiving object will iterate.

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to an initialized somf_TPriorityQueueIterator object.

Example

```
somf_TPriorityQueue pq;
Environment *ev;
somf_TPriorityQueueIterator itr;
ev = somGetGlobalEnvironment();
pq = somf_TPriorityQueueNew();
itr = somf_TPriorityQueueIteratorNew();
_somfTPriorityQueueIteratorInit(itr, ev, pq);
_somFree (pq);
_somFree (itr);
```

Original Class

somf_TPriorityQueueIterator

Related Information

None.

Chapter 19. somf_TSequence Class

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

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

Member Name:

tseq

Import Name:

GOSSOMUC

Base Class:

somf_TCollection

Metaclass:

SOMClass

Ancestor Classes:

somf_TCollection
somf_MCollectible
SOMObject

Subclasses:

None.

Types:

None.

Attributes:

None.

New Methods:

somfFirst
somfAfter
somfBefore
somfLast
somfOccurrencesOf
somfTSequenceInit

Overriding Methods:

somfAdd
somfRemove
somfRemoveAll
somfDeleteAll
somfCount
somfCreateIterator
somDefaultInit

somfAdd Method

Purpose

Adds an object to a given ordered collection. Every class that inherits from the somf_TSequence class must override this method for that class to work correctly.

Syntax

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

Parameters

receiver A pointer to an object of class somf_TSequence.

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

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

Restrictions and Limitations

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.

Returned Values

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 Method or somfIsSame Method would be TRUE.

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

Example

For examples of how this method looks when it is invoked, see somf_TDeque Class or somf_TSortedSequence Class.

Original Class

somf_TCollection (overridden here)

Related Information

None.

somfAfter Method

Purpose

Gets the object found after a given object *obj* in an ordered collection. Every class that inherits from the somf_TSequence class must override this method for that class to work correctly.

Syntax

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

Parameters

receiver A pointer to an object of class somf_TSequence.

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

obj A pointer to the somf_MCollectible object that is in front of the returned obj.

Restrictions and Limitations

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.

Returned Values

somf_MCollectible
A pointer to the somf_MCollectible object after obj.

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

Example

For examples of how this method is invoked, see somf_TDeque Class or somf_TSortedSequence Class.

Original Class

somf_TSequence

Related Information

somfBefore
somfFirst
somfLast

somfBefore Method

Purpose

Gets the object found before a given obj in an ordered collection. Every class that inherits from the somf_TSequence class must override this method for that class to work correctly.

Syntax

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

Parameters

receiver A pointer to an object of class somf_TSequence.

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

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

Restrictions and Limitations

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.

Returned Values

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

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

Example

For examples of how this method is invoked, see somf_TDeque Class or somf_TSortedSequence Class.

Original Class

somf_TSequence

Related Information

somfAfter
somfFirst
somfLast

somfCount Method

Purpose

Gets the number of objects in this ordered collection. Every class that inherits from the somf_TSequence class must override this method for that class to work correctly.

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

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

Syntax

```
long  somfCount ();
```

Parameters

receiver A pointer to an object of class somf_TSequence.

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

Restrictions and Limitations

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.

Returned Values

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

Example

For examples of how this method is invoked, see somf_TDeque Class or somf_TSortedSequence Class.

Original Class

somf_TCollection (overridden here)

Related Information

None.

somfCreateIterator Method

Purpose

Returns a new iterator that is suitable for iterating over the objects in an ordered collection. Every class that inherits from the somf_TSequence class must override this method for that class to work correctly.

Syntax

```
somf_TIterator  somfCreateIterator ();
```

Parameters

receiver A pointer to an object of class somf_TSequence.

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

Restrictions and Limitations

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.

Returned Values

This method returns a pointer to the new iterator.

Example

For examples of how this method is invoked, see somf_TDeque Class or somf_TSortedSequence Class.

Original Class

somf_TCollection (overridden here)

Related Information

None.

somfDeleteAll Method

Purpose

Removes all of the objects from an ordered collection and deallocates the storage that these objects might have owned. (That is, the destructor function is called for each object in the collection.) The method also deallocates the storage that these objects might have owned (that is, the destructor function is called for each object in the collection). Every class that inherits from the somf_TSequence class must override this method for that class to work correctly.

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

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

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

Syntax

```
void  somfDeleteAll ();
```

Parameters

receiver A pointer to an object of class somf_TSequence.

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

Restrictions and Limitations

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.

Returned Values

None.

Example

For examples of how this method is invoked, see somf_TDeque Class or somf_TSortedSequence Class.

Original Class

somf_TCollection (overridden here)

Related Information

None.

somfFirst Method

Purpose

Gets the first object in an ordered collection. Every class that inherits from the somf_TSequence class must override this method for that class to work correctly.

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

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

Syntax

```
somf_MCollectible  somfFirst ();
```

Parameters

receiver A pointer to an object of class somf_TSequence.

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

Returned Values

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

SOMF_NIL Nothing is in the collection.

Restrictions and Limitations

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.

Returned Values

None.

Example

For examples of how this method is invoked, see somf_TDeque Class or somf_TSortedSequence Class.

Original Class

somf_TSequence

Related Information

somfLast
somfAfter
somfBefore

somfLast Method

Purpose

Gets the last object in an ordered collection. Every class that inherits from the somf_TSequence class must override this method for that class to work correctly.

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

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

Syntax

```
somf_MCollectible  somfLast ();
```

Parameters

receiver A pointer to an object of class somf_TSequence.

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

Restrictions and Limitations

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.

Returned Values

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

SOMF_NIL Nothing is in the collection.

Example

For examples of how this method is invoked, see somf_TDeque Class or somf_TSortedSequence Class.

Original Class

somf_TSequence

Related Information

somfAfter
somfBefore
somfFirst

somfOccurrencesOf Method

Purpose

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

Syntax

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

Parameters

receiver A pointer to an object of class somf_TSequence.

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

obj A pointer to the somf_MCollectible object to look for in the collection.

Restrictions and Limitations

None.

Returned Values

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

Example

```
somf_TDeque dq;
<Your Class which inherits from somf_MCollectible> obj;
Environment *ev;
ev = somGetGlobalEnvironment();
dq = somf_TDequeNew();
obj = <Your Class which inherits from somf_MCollectible>New();
_somfAddFirst(dq, ev, obj);
somPrintf("\n There are %d OccurrencesOf obj\n",
                          _somfOccurrencesOf(dq, ev, obj));
_somFree (dq);
_somFree (obj);
```

Original Class

somf_TSequence

Related Information

None.

somfRemove Method

Purpose

Removes an object from an ordered collection. Every class that inherits from the somf_TSequence Class must override this method for that class to work correctly.

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

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

Syntax

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

Parameters

receiver A pointer to an object of class somf_TSequence.

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

obj A pointer to the somf_MCollectible object to be removed from the collection.

Restrictions and Limitations

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.

Returned Values

somf_MCollectible
A pointer to the object which was removed.

SOMF_NIL The object was not found.

Example

For examples of how this method is invoked, see somf_TDeque Class or somf_TSortedSequence Class.

Original Class

somf_TCollection (overridden here)

Related Information

somfRemoveAll

somfRemoveAll Method

Purpose

Removes all of the objects from an ordered collection. Every class that inherits from the somf_TSequence class must override this method for that class to work correctly.

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

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

Syntax

```
void somfRemoveAll ();
```

Parameters

receiver A pointer to an object of class somf_TSequence.

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

Restrictions and Limitations

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.

Returned Values

None.

Example

For examples of how this method is invoked, see somf_TDeque Class or somf_TSortedSequence Class.

Original Class

somf_TCollection (overridden here)

Related Information

somfRemove

somfTSequenceInit Method

Purpose

Initializes a new ordered collection of class somf_TSequence, given a comparison method for the collection to use. The method also establishes the comparison method that the new ordered collection will use to compare current potential objects for the collection, as determined by the *testfn* argument.

Note: You cannot override this method.

Syntax

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

Parameters

receiver A pointer to an object of class somf_TSequence.

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

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

This argument should always be set to either

`somf_MCollectibleClassData.somfIsSame or somf_MCollectibleClassData.somfIsEqual.`

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

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to an initialized somf_TSequence object.

Original Class

somf_TSequence

Related Information

None.

Chapter 20. somf_TSequenceIterator Class

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

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

Member Name:

tseqitr

Import Name:

GOSSOMUC

Base Class:

somf_TIterator

Metaclass:

SOMClass

Ancestor Classes:

somf_TIterator
SOMObject

Subclasses:

None.

Types:

None.

Attributes:

None.

New Methods:

somfLast
somfPrevious

Overriding Methods:

somfFirst
somfNext
somfRemove

somfFirst Method

Purpose

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

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

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

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

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

Syntax

somf_MCollectible **somfFirst** ();

Parameters

receiver A pointer to an object of class somf_TSequenceIterator.

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

Restrictions and Limitations

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.

Returned Values

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

Example

For examples of how this method is invoked, see somf_TDequeIterator Class or somf_TSortedSequenceIterator Class.

Original Class

somf_TIterator (overridden here)

Related Information

somfNext

somfLast Method

Purpose

Gets the last object in an ordered collection. Every class that inherits from the somf_TSequenceIterator class must override this method for that class to work correctly.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. somfLast is a method name declared in multiple parents (for example: somf_TSequenceIterator, somf_TSequence). You will probably have to fully qualify the name of the method (for example: somf_TSortedSequenceIterator_somfLast). This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

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

Syntax

```
somf_MCollectible  somfLast ();
```

Parameters

receiver A pointer to an object of class somf_TSequenceIterator.

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

Restrictions and Limitations

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.

Returned Values

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

Example

For examples of how this method is invoked, see somf_TDequeIterator Class or somf_TSortedSequenceIterator Class.

Original Class

somf_TSequenceIterator

Related Information

somfPrevious

somfNext Method

Purpose

Gets the next object in an ordered collection. The method also returns a pointer to the next object, if found. Objects are retrieved in an order that reflects the "orderedness" of the collection (or the lack of ordering on the collection elements).

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

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

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

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

Syntax

somf_MCollectible **somfNext** ();

Parameters

receiver A pointer to an object of class somf_TSequenceIterator.

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

Restrictions and Limitations

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.

Returned Values

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

SOMF_NIL The end of the collection has been reached.

Example

For examples of how this method is invoked, see somf_TDequeIterator Class or somf_TSortedSequenceIterator Class.

Original Class

somf_TIterator (overridden here)

Related Information

somfFirst

somfPrevious Method

Purpose

Gets the previous object in an ordered collection. Every class that inherits from the somf_TSequenceIterator class must override this method for that class to work correctly.

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

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

Syntax

somf_MCollectible **somfPrevious** ();

Parameters

receiver A pointer to an object of class somf_TSequenceIterator.

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

Restrictions and Limitations

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.

Returned Values

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

SOMF_NIL The beginning of the collection has been reached.

Example

For examples of how this method is invoked, see somf_TDequeIterator Class or somf_TSortedSequenceIterator Class.

Original Class

somf_TSequenceIterator

Related Information

somfLast

somfRemove Method

Purpose

Removes the current object from an ordered collection.

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

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

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

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

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

Syntax

```
void  somfRemove ();
```

Parameters

receiver A pointer to an object of class somf_TSequenceIterator.

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

Restrictions and Limitations

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.

Returned Values

None.

Example

For examples of how this method is invoked, see somf_TDequeIterator Class or somf_TSortedSequenceIterator Class.

Original Class

somf_TIterator (overridden here)

Related Information

None.

Chapter 21. somf_TSet Class

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

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

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

The somf_TSet class uses somfIsEqual as the default comparison function. If key1="Bart" and key2="Bart", then key1 and key2 are equal. If you do not want to use this method to equate entries, use an initialization methods to change to somfIsSame.

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

Member Name:

tset

Import Name:

GOSSOMUC

Base Class:

somf_TCollection

Metaclass:

SOMClass

Ancestor Classes:

somf_TCollection
somf_MCollectible
SOMObject

Subclasses:

None.

Types:

None.

Attributes:

None.

New Methods:

somfDifferenceS
somfDifferenceSS
somfIntersectionS
somfIntersectionSS
somfUnionS
somfUnionSS
somfXorS
somfXorSS
somfSetHashFunction
somfGetHashFunction
somfRehash
somfAssign
somfTSetInitFL
somfTSetInitF
somfTSetInitLF
somfTSetInitL
somfTSetInitS

Overriding Methods:

somDefaultInit
somDestruct
somfAdd
somfRemove
somfRemoveAll
somfDeleteAll
somfCount
somfMember
somfCreateIterator

somfAdd Method

Purpose

Adds an object to a given set.

Syntax

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

Parameters

receiver A pointer to an object of class somf_TSet.

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

obj A pointer to the somf_MCollectible Class object that will be added to the set.

Restrictions and Limitations

None.

Returned Values

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

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

Example

```
somf_TSet s;
<Your class which inherits from somf_MCollectible> obj;
Environment *ev;
ev = somGetGlobalEnvironment();
s = somf_TSetNew();
obj = <Your class which inherits from somf_MCollectible>New();
/* Add obj to s */
if (_somfAdd(s, ev, obj) != SOMF_NIL)
    somPrintf("\n problem adding obj to s\n");
_somFree (s);
```

Original Class

somf_TCollection (overridden here)

Related Information

None.

somfAssign Method

Purpose

Assigns a set as being equal to a given source set. That is, the method sets/resets the instance variables of the receiver to the values of the source. This operation is logically equivalent to using the equal (=) operator.

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

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

Syntax

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

Parameters

receiver A pointer to an object of class somf_TSet.

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

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

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_TSet s1;
somf_TSet s2;
Environment *ev;
ev = somGetGlobalEnvironment();
s1 = somf_TSetNew();
s2 = somf_TSetNew();
/* Add som objects to s1 */
/* Assign s2 = s1 */
somf_TSet_somfAssign(s2, ev, s1);
_somFree (s1);
_somFree (s2);
```

Original Class

somf_TSet

Related Information

None.

somfCount Method

Purpose

Gets the number of objects in a given set.

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

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

Syntax

```
long somfCount ();
```

Parameters

receiver A pointer to an object of class somf_TSet.

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

Restrictions and Limitations

None.

Returned Values

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

Example

```
somf_TSet s;
Environment *ev;
ev = somGetGlobalEnvironment();
s = somf_TSetNew();
somPrintf("\n Count of s= %d\n", _somfCount(s,ev));
_somFree (s);
```

Original Class

somf_TCollection (overridden here)

Related Information

None.

somfCreateIterator Method

Purpose

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

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

Syntax

somf_TIterator **somfCreateIterator** ();

Parameters

receiver A pointer to an object of class somf_TSet.

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

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to the new iterator.

Example

```
somf_TSet s;
Environment *ev;
somf_TSetIterator itr;
ev = somGetGlobalEnvironment();
s = somf_TSetNew();
itr = (somf_TSetIterator*) _somfCreateIterator(s,ev);
_somFree (s);
_somFree (itr);
```

Original Class

somf_TCollection (overridden here)

Related Information

None.

somfDeleteAll Method

Purpose

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

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

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

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

Syntax

void **somfDeleteAll** ();

Parameters

receiver A pointer to an object of class somf_TSet.

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

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_TSet s;
Environment *ev;
ev = somGetGlobalEnvironment();
s = somf_TSetNew();
/* Add some objects to s */
/* Remove all of the objects from s AND DELETE THEM */
_somfDeleteAll(s,ev);
_somFree (s);
```

Original Class

somf_TCollection (overridden here)

Related Information

None.

somfDifferenceS Method

Purpose

Determines the elements of a given set that do not appear in another specified set, and modifies the first set to contain only those different elements. The set used as the receiving object is destructively modified to contain only those elements that do not appear in set1.

Syntax

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

Parameters

receiver A pointer to an object of class somf_TSet.

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

set1 A pointer to the set that the receiving object will be compared against.

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_TSet s1;
somf_TSet s2;
Environment *ev;
ev = somGetGlobalEnvironment();
s1 = somf_TSetNew();
s2 = somf_TSetNew();
/* Find the differences between s1 and s2, and put in s1 */
_somfDifferenceS(s1, ev, s2);
_somFree (s1);
_somFree (s2);
```

Original Class

somf_TSet

Related Information

somfDifferenceSS

somfDifferenceSS Method

Purpose

Determines the elements of a given set that do not appear in another specified set, and places those different elements in a third set. Those elements that do not appear in set are then placed in set *resultSet*. The original receiving-object set remains unchanged.

Syntax

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

Parameters

receiver A pointer to an object of class somf_TSet.

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

set1 A pointer to the set that the receiving object set will be compared against.

resultSet A pointer to the set containing the results of the operation.

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_TSet s1;
somf_TSet s2;
somf_TSet s3;
Environment *ev;
ev = somGetGlobalEnvironment();
s1 = somf_TSetNew();
s2 = somf_TSetNew();
s3 = somf_TSetNew();
/* Find the differences between s1 and s2, and put in s3 */
_somfDifferenceSS(s1, ev, s2, s3);
_somFree (s1);
_somFree (s2);
_somFree (s3);
```

Original Class

somf_TSet

Related Information

somfDifferenceS

somfGetHashFunction Method

Purpose

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

C cannot handle methods from different classes having the same name when they inherit the name from different parents. If somf_TSet is used with a child of somf_THashTable or somf_TDictionary, then the name of the method will have to be fully qualified (example: somf_TSet_somfGetHashFunction). This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

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

Syntax

somf_MCollectibleHashFn
somfGetHashFunction ();

Parameters

receiver A pointer to an object of class somf_TSet.

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

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to the hash function.

Example

```
somf_TSet s;
Environment *ev;
ev = somGetGlobalEnvironment();
s = somf_TSetNew();
if ((somf_TSet_somfGetHashFunction(s,ev)) !=
              somf_MCollectibleClassData.somfHash)
    somPrintf("\n What Hash Function are we using?\n");
_somFree (s);
```

Original Class

somf_TSet

Related Information

somfSetHashFunction

somfIntersectionS Method

Purpose

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

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

Syntax

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

Parameters

receiver A pointer to an object of class somf_TSet.

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

set1 A pointer to the set that the receiving object will be compared against.

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_TSet s1;
somf_TSet s2;
Environment *ev;
ev = somGetGlobalEnvironment();
s1 = somf_TSetNew();
s2 = somf_TSetNew();
/* Find the intersection between s1 and s2, and put in s1 */
_somfIntersectionS(s1, ev, s2);
_somFree (s1);
_somFree (s2);
```

Original Class

somf_TSet

Related Information

somfIntersectionSS

somfIntersectionSS Method

Purpose

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

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

Syntax

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

Parameters

receiver A pointer to an object of class somf_TSet.

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

set1 A pointer to the set this instance will be compared against.

resultSet A pointer to the set containing the results of the operation.

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_TSet s1;
somf_TSet s2;
somf_TSet s3;
Environment *ev;
ev = somGetGlobalEnvironment();
s1 = somf_TSetNew();
s2 = somf_TSetNew();
s3 = somf_TSetNew();
/* Find the intersection between s1 and s2, and put in s3 */
_somfIntersectionSS(s1, ev, s2, s3);
_somFree (s1);
_somFree (s2);
_somFree (s3);
```

Original Class

somf_TSet

Related Information

somfIntersectionS

somfMember Method

Purpose

Gets an object from a set.

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

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

Syntax

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

Parameters

receiver A pointer to an object of class somf_TSet.

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

obj A pointer to the somf_MCollectible that may or may not be a member of the Collection.

Restrictions and Limitations

None.

Returned Values

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

SOMF_NIL The object was not found.

Example

```
somf_TSet s;
<Your Class which inherits from somf_MCollectible> obj;
Environment *ev;
ev = somGetGlobalEnvironment();
s = somf_TSetNew();
obj = <Your Class which inherits from somf_MCollectible>New();
_somfAdd(s, ev, obj);
if (_somfMember(s, ev, obj) != SOMF_NIL)
    somPrintf("\n obj is a Member\n");
else
    somPrintf("\n ERROR: obj should be a Member\n");
_somFree (s);
_somFree (obj);
```

Original Class

somf_TCollection (overridden here)

Related Information

None.

somfRehash Method

Purpose

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

Note: You cannot override this method.

Syntax

```
void somfRehash ();
```

Parameters

receiver A pointer to an object of class somf_TSet.

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

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_TSet s;
Environment *ev;
ev = somGetGlobalEnvironment();
s = somf_TSetNew();
_somfRehash(s,ev);
_somFree (s);
```

Original Class

somf_TSet

Related Information

None.

somfRemove Method

Purpose

Removes an object from a given set.

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

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

Syntax

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

Parameters

receiver A pointer to an object of class somf_TSet.

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

obj A pointer to the somf_MCollectible object to be removed from the set.

Restrictions and Limitations

None.

Returned Values

somf_MCollectible
A pointer to the object that was removed.

SOMF_NIL The object was not found.

Example

```
somf_TSet s;
<Your Class which inherits from somf_MCollectible> obj;
Environment *ev;
ev = somGetGlobalEnvironment();
s = somf_TSetNew();
obj = <Your Class which inherits from somf_MCollectible>New();
_somfAdd(s, ev, obj);
if (somf_TSet_somfRemove(s, ev, obj) == SOMF_NIL)
    somPrintf("\n problem removing obj from s\n");
_somFree (s);
_somFree (obj);
```

Original Class

somf_TCollection (overridden here)

Related Information

somfRemoveAll

somfRemoveAll Method

Purpose

Removes all of the objects from a given set.

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

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

Syntax

void **somfRemoveAll** ();

Parameters

receiver A pointer to an object of class somf_TSet.

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

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_TSet s;
Environment *ev;
ev = somGetGlobalEnvironment();
s = somf_TSetNew();
/* Remove All of the objects in s */
_somfRemoveAll(s,ev);
_somFree (s);
```

Original Class

somf_TCollection (overridden here)

Related Information

somfRemove

somfSetHashFunction Method

Purpose

Sets the hash function of a set.

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

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

Syntax

```
void  somfSetHashFunction(
    in somf_MCollectibleHashFn fn);
```

Parameters

receiver A pointer to an object of class somf_TSet.

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

fn A function pointer specifying a somfHash Method type function.

This argument should always be set to

```
somf_MCollectibleClassData.somfHash
```

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

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_TSet s;
Environment *ev;
ev = somGetGlobalEnvironment();
s = somf_TSetNew();
somf_TSet_somfSetHashFunction(s, ev,
                    somf_MCollectibleClassData.somfHash);
_somFree (s);
```

Original Class

somf_TSet

Related Information

somfGetHashFunction

somfTSetInitF Method

Purpose

Initializes a new set, given its comparison test method. The method assumes a default number of objects as the set size.

Note: You cannot override this method.

Syntax

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

Parameters

receiver A pointer to an object of class somf_TSet.

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

testfn A method pointer specifying either a somfIsEqual or a 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 Class. The somf_TSet object will use this pointer to access the somfIsSame Method or somfIsEqual Method that was declared and defined in the object being inserted into, or removed from, the somf_TSet object.

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to an initialized somf_TSet object.

Example

```
somf_TSet s2;
Environment *ev;
ev = somGetGlobalEnvironment();
s2 = somf_TSetNew();
_somfTSetInitF(s2, ev, somf_MCollectibleClassData.somfIsEqual);
_somFree (s2);
```

Original Class

somf_TSet

Related Information

somfTSetInitFL
somfTSetInitLF
somfTSetInitL
somfTSetInitS

somfTSetInitFL Method

Purpose

Initializes a new set, given the comparison test method and the initial set size. This method is equivalent ot the somfTSetInitFL method.

Note: You cannot override this method.

Syntax

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

Parameters

receiver A pointer to an object of class somf_TSet.

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

testfn A method pointer specifying either a somfIsEqual or a 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 Class. The somf_TSet object will use this pointer to access the somfIsSame Method or somfIsEqual Method that was declared and defined in the object being inserted into, or removed from, the somf_TSet object.

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

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to an initialized somf_TSet object.

Example

```
somf_TSet s1;
Environment *ev;
ev = somGetGlobalEnvironment();
s1 = somf_TSetNew();
_somfTSetInitFL(s1, ev, somf_MCollectibleClassData.somfIsEqual, 8);
_somFree (s1);
```

Original Class

somf_TSet

Related Information

somfTSetInitLF
somfTSetInitF
somfTSetInitL
somfTSetInitS

somfTSetInitL Method

Purpose

Initializes a new set, given the initial set size. The method assumes the somf_TSet class's default comparison test function of somfIsEqual.

Note: You cannot override this method.

Syntax

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

Parameters

receiver A pointer to an object of class somf_TSet.

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

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

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to an initialized somf_TSet.

Example

```
somf_TSet s4;
Environment *ev;
ev = somGetGlobalEnvironment();
s4 = somf_TSetNew();
_somfTSetInitL(s4, ev, 8);
_somFree (s4);
```

Original Class

somf_TSet

Related Information

somfTSetInitFL
somfTSetInitLF
somfTSetInitF
somfTSetInitS

somfTSetInitLF Method

Purpose

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

Note: You cannot override this method.

Syntax

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

Parameters

receiver A pointer to an object of class somf_TSet.

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

testfn A method pointer specifying either a somfIsEqual or a 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 Class. The somf_TSet object will use this pointer to access the somfIsSame Method or somfIsEqual Method that was declared and defined in the object being inserted into, or removed from, the somf_TSet object.

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to an initialized somf_TSet object.

Example

```
somf_TSet s3;
Environment *ev;
ev = somGetGlobalEnvironment();
s3 = somf_TSetNew();
_somfTSetInitLF(s3, ev, 8,
                somf_MCollectibleClassData.somfIsEqual);
_somFree (s3);
```

Original Class

somf_TSet

Related Information

somfTSetInitFL
somfTSetInitF
somfTSetInitL
somfTSetInitS

somfTSetInitS Method

Purpose

Initializes a new set, establishing it as equal to another given set. The method also establishes the new set as equal to the specified source set. This implies that the instance data of the new set will be equal to those of the source set.

Note: You cannot override this method.

Syntax

```
somf_TSet  somfTSetInitS(
    in somf_TSet s);
```

Parameters

receiver A pointer to an object of class somf_TSet.

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

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

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to an initialized somf_TSet object.

Example

```
somf_TSet s4;
somf_TSet s5;
Environment *ev;
ev = somGetGlobalEnvironment();
s4 = somf_TSetNew();
s5 = somf_TSetNew();
_somfTSetInitS(s5, ev, s4);
_somFree (s4);
_somFree (s5);
```

Original Class

somf_TSet

Related Information

somfTSetInitFL
somfTSetInitLF
somfTSetInitF
somfTSetInitL

somfUnionS Method

Purpose

Gets those elements that are members of either a given set or another set, *set1*, and modifies the first set to contain all elements from both sets. The set used as the receiving object is then destructively modified to contain all of those elements from both sets.

Syntax

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

Parameters

receiver A pointer to an object of class somf_TSet.

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

set1 A pointer to the set that the receiving object will be compared against.

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_TSet s1;
somf_TSet s2;
Environment *ev;
ev = somGetGlobalEnvironment();
s1 = somf_TSetNew();
s2 = somf_TSetNew();
/* Find the union between s1 and s2, and put it in s1 */
_somfUnionS(s1, ev, s2);
_somFree (s1);
_somFree (s2);
```

Original Class

somf_TSet

Related Information

somfUnionSS

somfUnionSS Method

Purpose

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

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

Syntax

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

Parameters

receiver A pointer to an object of class somf_TSet.

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

set1 A pointer to the set that the receiving object will be compared against.

resultSet A pointer to the set containing the results of the operation.

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_TSet s1;
somf_TSet s2;
somf_TSet s3;
Environment *ev;
ev = somGetGlobalEnvironment();
s1 = somf_TSetNew();
s2 = somf_TSetNew();
s3 = somf_TSetNew();
/* Find the union between s1 and s2, and put it in s3 */
_somfUnionSS(s1, ev, s2, s3);
_somFree (s1);
_somFree (s2);
_somFree (s3);
```

Original Class

somf_TSet

Related Information

somfUnionS

somfXorS Method

Purpose

Determines a set wherein each member is an element either of a given set or of another set *set1*, but not of both, and modifies the first set to contain the elements of the new set. The receiving object set is then modified to contain all of the elements of the newly determined set.

Syntax

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

Parameters

receiver A pointer to an object of class somf_TSet.

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

set1 A pointer to the set that the receiving object will be compared against.

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_TSet s1;
somf_TSet s2;
Environment *ev;
ev = somGetGlobalEnvironment();
s1 = somf_TSetNew();
s2 = somf_TSetNew();
/* Find the exclusive or of s1 and s2, and put it in s1 */
_somfXorS(s1, ev, s2);
_somFree (s1);
_somFree (s2);
```

Original Class

somf_TSet

Related Information

somfXorSS

somfXorSS Method

Purpose

Determines a set where each member is an element either of a given set or of another set *set1*, but not of both, and places all of those elements in a third set. All elements of the newly determined set are then placed in set resultSet. The receiving-object set and set1 remain unchanged.

Syntax

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

Parameters

receiver A pointer to an object of class somf_TSet.

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

set1 A pointer to the set that the receiving object will be compared against.

resultSet A pointer to the set containing the results of the operation.

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_TSet s1;
somf_TSet s2;
somf_TSet s3;
Environment *ev;
ev = somGetGlobalEnvironment();
s1 = somf_TSetNew();
s2 = somf_TSetNew();
s3 = somf_TSetNew();
/* Find the exclusive or of s1 and s2, and put it in s3 */
_somfXorSS(s1, ev, s2, s3);
_somFree (s1);
_somFree (s2);
_somFree (s3);
```

Original Class

somf_TSet

Related Information

somfXorS

Chapter 22. somf_TSetIterator Class

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

Member Name:

tsetitr

Import Name:

GOSSOMUC

Base Class:

somf_TIterator

Metaclass:

SOMClass

Ancestor Classes:

somf_TIterator
SOMObject

Subclasses:

None.

Types:

None.

Attributes:

None.

New Methods:

somfTSetIteratorInit

Overriding Methods:

somDestruct
somfNext
somfFirst
somfRemove

somfFirst Method

Purpose

Resets the iterator and returns the first element of a set. somfFirst resets the iterator to the beginning of the set even if other operations on the collection cause the iterator to be invalidated. In the second case, the method revalidates the iterator.

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

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

Syntax

somf_MCollectible **somfFirst** ();

Parameters

receiver A pointer to an object of class somf_TSetIterator.

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

Restrictions and Limitations

None.

Returned Values

somf_MCollectible
A pointer to the first somf_MCollectible object in the set.

SOMF_NIL Is returned if the collection is empty.

Example

```
somf_TSet s;
Environment *ev;
somf_TSetIterator itr;
somf_MCollectible itrobj;
ev = somGetGlobalEnvironment();
s = somf_TSetNew();
itr = somf_TSetIteratorNew();
_somfTSetIteratorInit(itr, ev, s);
/* Add some object to s */
/* Iterate through the TSet */
itrobj = somf_TSetIterator_somfFirst(itr,ev);
while (itrobj != SOMF_NIL)
{
    /* Do something with itrobj */
    itrobj = _somfNext(itr,ev);
}
_somFree (s);
_somFree (itr);
```

Original Class

somf_TIterator

Related Information

somfNext

somfNext Method

Purpose

Gets the next object in a set. Objects are retrieved in an order reflecting the "ordered-ness" of the set (or the lack of ordering on the set elements).

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

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

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

Syntax

somf_MCollectible **somfNext** ();

Parameters

receiver A pointer to an object of class somf_TSetIterator.

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

Restrictions and Limitations

None.

Returned Values

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

SOMF_NIL The end of the set has been reached.

Example

```
somf_TSet s;
Environment *ev;
somf_TSetIterator itr;
somf_MCollectible itrobj;
ev = somGetGlobalEnvironment();
s = somf_TSetNew();
itr = somf_TSetIteratorNew();
_somfTSetIteratorInit(itr, ev, s);
/* Add some object to s */
/* Iterate through the TSet */
itrobj = somf_TSetIterator_somfFirst(itr,ev);
while (itrobj != SOMF_NIL)
{
    /* Do something with itrobj */
    itrobj = _somfNext(itr,ev);
}
_somFree (s);
_somFree (itr);
```

Original Class

somf_TIterator

Related Information

somfFirst

somfRemove Method

Purpose

Removes the current object, the one just returned by somfFirst or somfNext, from a set.

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

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

C cannot handle methods from different classes having the same name when they inherit the name from different parents. somfRemove is a method name declared in multiple parents (example: somf_TCollection, somf_THashTable, somf_TIterator). You will probably have to fully qualify the method name. This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

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

Syntax

```
void  somfRemove ();
```

Parameters

receiver A pointer to an object of class somf_TSetIterator.

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

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_TSet s;
Environment *ev;
somf_TSetIterator itr;
ev = somGetGlobalEnvironment();
s = somf_TSetNew();
itr = somf_TSetIteratorNew();
_somfTSetIteratorInit(itr, ev, s);
/* Remove the first object in s */
/* Iterate through the TSet */
somf_TSetIterator_somfFirst(itr,ev);
somf_TSetIterator_somfRemove(itr,ev);
_somFree (s);
_somFree (itr);
```

Original Class

somf_TIterator

Related Information

None.

somfTSetIteratorInit Method

Purpose

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

This is one of two ways to initialize a somf_TSetIterator to point to an instance of a somf_TSet set. The other way is to use the somf_TSet class's somfCreateIterator Method.

Note: You cannot override this method.

Syntax

```
somf_TSetIterator
somfTSetIteratorInit(
    in somf_TSet h);
```

Parameters

receiver A pointer to an object of class somf_TSetIterator.

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

h A pointer to the set that the receiving object will iterate over.

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to an initialized somf_TSetIterator object.

Example

```
somf_TSet s;
Environment *ev;
somf_TSetIterator itr;
ev = somGetGlobalEnvironment();
s = somf_TSetNew();
itr = somf_TSetIteratorNew();
_somfTSetIteratorInit(itr, ev, s);
_somFree (s);
_somFree (itr);
```

Original Class

somf_TSetIterator

Related Information

None.

Chapter 23. somf_TSortedSequence Class

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

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

All somf_MOrderableCollectible objects inserted into a somf_TSortedSequence collection should override somfIsEqual, somfIsLessThan, somfIsGreaterThan and somfHash.

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

Member Name:

tss

Import Name:

GOSSOMUC

Base Class:

somf_TSequence

Metaclass:

SOMClass

Ancestor Classes:

somf_TSequence
somf_TCollection
somf_MCollectible
SOMObject

Subclasses:

None.

Types:

None.

Attributes:

None.

New Methods:

somfAssign
somfCreateSequenceIterator
somfCreateSortedSequenceNode
somfGetSequencingFunction
somfSetSequencingFunction
somfTSortedSequenceInitF
somfTSortedSequenceInitS

Overriding Methods;

somDefaultInit
somDestruct
somfAdd
somfAfter
somfBefore
somfCount
somfCreateIterator
somfDeleteAll
somfFirst
somfLast
somfMember
somfOccurrencesOf
somfRemove
somfRemoveAll

somfAdd Method

Purpose

Adds an obj to a sorted sequence.

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

Syntax

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

Parameters

receiver A pointer to an object of class somf_TSortedSequence

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

obj A pointer to an somf_MCollectible that will be added to this instance.

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to the somf_MCollectible object added.

Example

```
somf_TSortedSequence ss;
<Your class inheriting from somf_MOrderableCollectible> obj;
Environment *ev;
ev = somGetGlobalEnvironment();
ss = somf_TSortedSequenceNew();
obj =
   <Your class inheriting from somf_MOrderableCollectible>New();
/* Add obj to ss */
_somfAdd(ss, ev, obj);
_somFree (ss);
```

Original Class

somf_TCollection

Related Information

None.

somfAfter Method

Purpose

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

Syntax

```
somf_MCollectible somfAfter(
    in somf_MCollectible obj);
```

Parameters

receiver A pointer to an object of class somf_TSortedSequence.

ev A pointer to the Environment structure for the calling method.

obj A pointer to the somf_MCollectible object that precedes the returned *obj*.

Restrictions and Limitations

None.

Returned Values

somf_MCollectible
A pointer to the somf_MCollectible object after *obj*.

SOMF_NIL The designated *obj* is the last object in this collection, or not found.

Example

```
somf_TSortedSequence ss;
<Your class inheriting from somf_MOrderableCollectible> obj;
<Your class inheriting from somf_MOrderableCollectible> objptr;
Environment *ev;
ev = somGetGlobalEnvironment();
ss = somf_TSortedSequenceNew();
obj = <Your class that inherits from \n
somf_MOrderableCollectible>New();
/* Determine what object comes after obj */
objptr =
       (<Your class inheriting from somf_MOrderableCollectible>*)
   _somfAfter(ss,ev,obj);
_somFree (ss);
```

Original Class

somf_TSortedSequence (overridden here)

Related Information

somfBefore
somfFirst
somfLast

somfAssign Method

Purpose

Assigns a sorted sequence as equal to a given source sorted sequence. That is, the method sets or resets the instance variables of the receiver to the values of the source. This operation is logically equivalent to using the equals (=) operator.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Thus, if any child of somf_TSortedSequence is used with any other main collection class, then the name of the method will have to be fully qualified (example: somf_TSortedSequence_somfAssign). This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as

```
d->somfAssign(ev, d2);
```

Syntax

```
void somfAssign(
    in somf_TSortedSequences);
```

Parameters

receiver A pointer to an object of class somf_TSortedSequence.

ev A pointer to the Environment structure for the calling method.

s A pointer to the sorted sequence to which the receiver will be equal.

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_TSortedSequence s1;
somf_TSortedSequence s2;
Environment *ev;
ev = somGetGlobalEnvironment();
s1 = somf_TSortedSequenceNew();
s2 = somf_TSortedSequenceNew();
/* Add som objects to s1 */
/* Assign s2 = s1 */
somf_TSortedSequence_somfAssign(s2, ev, s1);
_somFree (s1);
_somFree (s2);
```

Original Class

somf_TSortedSequence

Related Information

None.

somfBefore Method

Purpose

Returns the object found before a given object in a sorted sequence.

Syntax

```
somf_MCollectible somfBefore(
    in somf_MCollectible obj);
```

Parameters

receiver A pointer to an object of class somf_TSortedSequence.

ev A pointer to the Environment structure for the calling method.

obj A pointer to the somf_MCollectible object that is behind the returned *obj*.

Restrictions and Limitations

None.

Returned Values

somf_MCollectible
A pointer to the somf_MCollectible object that precedes *obj*.

SOMF_NIL The designated *obj* is the first object in the sequence, or not found.

Example

```
somf_TSortedSequence ss;
<Your class inheriting from somf_MOrderableCollectible> obj;
<Your class inheriting from somf_MOrderableCollectible> objptr;
Environment *ev;
ev = somGetGlobalEnvironment();
ss = somf_TSortedSequenceNew();
obj =
  <Your class inheriting from somf_MOrderableCollectible>New();
/* Determine what object comes before obj */
objptr =
  (<Your class inheriting from somf_MOrderableCollectible>*)
   _somfBefore(ss,ev,obj);
_somFree (ss);
```

Original Class

somf_TSequence

Related Information

somfAfter
somfFirst
somfLast

somfCount Method

Purpose

The somfCount method determines the number of objects in the sorted sequence given as the receiving object, and returns that number.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Thus, if any child of somf_TCollection is used with a child of somf_THashTable, then the name of the method will have to be fully qualified (for example, somf_TDictionary_somfCount). This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

```
d->somfCount(ev);
```

Syntax

```
long  somfCount ();
```

Parameters

receiver A pointer to an object of class somf_TSorted Sequence.

ev A pointer to the Environment structure for the calling method.

Restrictions and Limitations

None.

Returned Values

This method returns the number of objects in this sorted sequence.

Example

```
somf_TSortedSequence ss;
Environment *ev;
ev = somGetGlobalEnvironment();
ss = somf_TSortedSequenceNew();
somPrintf("\n Count of ss= %d\n", _somfCount(ss,ev));
_somFree (ss);
```

Original Class

somf_TCollection (overridden here)

Related Information

None.

somfCreateIterator Method

Purpose

Returns a new iterator that is suitable for iterating over the objects in a sorted sequence.

Note: This is one of three ways to initialize a somf_TSortedSequenceIterator to point to an instance of a somf_TSortedSequence. One other way is to use the initializer method of the somf_TSortedSequenceIterator class. The final way is to use the somf_TSortedSequence class's somfCreateSequenceIterator method.

Syntax

```
somf_TIterator  somfCreateIterator ();
```

Parameters

receiver A pointer to an object of class somf_TSortedSequence.

ev A pointer to the Environment structure for the calling method.

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to the new iterator.

Example

```
somf_TSortedSequence ss;
Environment *ev;
somf_TSortedSequenceIterator itr;
ev = somGetGlobalEnvironment();
ss = somf_TSortedSequenceNew();
itr = (somf_TSortedSequenceIterator*) _somfCreateIterator(ss,ev);
_somFree (ss);
_somFree (itr);
```

Original Class

somf_TCollection

Related Information

somfCreateSequenceIterator

somfCreateSequenceIterator Method

Purpose

Returns a new iterator that is suitable for iterating over the objects in a sorted sequence.

Note: This is one of three ways to initialize a somf_TSortedSequenceIterator Class to point to an instance of a somf_TSortedSequence. One other way is to use the initializer method for the somf_TSortedSequenceIterator class. The final way is to use the somf_TSortedSequence class's somfCreateIterator.

This method is virtually identical to the somfCreateIterator method; thus, you could use either one. The only difference between the methods is the indicated type of their return value: the current method returns a somf_TSortedSequenceIterator object, whereas the somfCreateIterator method returns a somf_TIterator object. In reality, however, both methods return an instance of a somf_TSortedSequenceIterator that has been typed correctly.

Syntax

somf_TSequenceIterator
somfCreateSequenceIterator ();

Parameters

receiver A pointer to an object of class somf_TSortedSequence.

ev A pointer to the Environment structure for the calling method.

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to the new sorted-sequence iterator.

Example

```
somf_TSortedSequence ss;
Environment *ev;
somf_TSortedSequenceIterator itr;
ev = somGetGlobalEnvironment();
ss = somf_TSortedSequenceNew();
itr = (somf_TSortedSequenceIterator*)
                        _somfCreateSequenceIterator(ss,ev);
_somFree (ss);
_somFree (itr);
```

Original Class

somf_TSortedSequence

Related Information

somfCreateIterator

somfCreateSortedSequenceNode Method

Purpose

Creates a new somf_TSortedSequenceNode in a somf_TSortedSequence collection, given a key to the new node and its left and right children. The method call also specifies a somf_MOrderableCollectible object to be used as the key to the new node, as well as two somf_TSortedSequenceNode objects to be used as the left and right children of the new node.

If you create a new class that inherits from the somf_TSortedSequence class, you might consider overriding this method in order to customize how an instance of your new class creates a new node.

Syntax

```
somf_TSortedSequenceNode
somfCreateSortedSequenceNode(
    in somf_TSortedSequenceNode n1,
    in somf_MOrderableCollectible
    obj,
    in somf_TSortedSequenceNode n2);
```

Parameters

receiver A pointer to an object of class somf_TSortedSequence.

ev A pointer to the Environment structure for the calling method.

n1 A pointer to the left child of the new somf_TSortedSequenceNode object.

ob A pointer to the key of the new somf_TSortedSequenceNode object.

n2 A pointer to the right child of the new somf_TSortedSequenceNode object.

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to the new somf_TSortedSequenceNode created.

Example

None.

Original Class

somf_TSortedSequence

Related Information

None.

somfDeleteAll Method

Purpose

Removes all objects from a sorted sequence and deallocates the storage that these objects might have owned. The destructor function is called for each object in the collection.

Be careful with somfDeleteAll. Since a collection only contains pointers to objects (rather than the objects themselves), somfDeleteAll can cause a problem if a pointer to an object appears more than once. For example, if multiple pointers to "A" exists, or if a single pointer to "A" is in the collection multiple times, the behavior of the code is undefined, because it will try to delete "A" multiple times. If you think there is a chance that an object could appear in the collection more than once, you should consider using somfRemoveAll Method to remove the objects from the collection and deleting them some other way.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Thus, if any child of somf_TCollection is used with somf_THashTable, then the name of the method will have to be fully qualified (example: somf_TDictionary_somfDeleteAll). This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

```
d->somfDeleteAll(ev);
```

Syntax

void **somfDeleteAll** ();

Parameters

receiver A pointer to an object of class somf_TSortedSequence.

ev A pointer to the Environment structure for the calling method.

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_TSortedSequence ss;
Environment *ev;
ev = somGetGlobalEnvironment();
ss = somf_TSortedSequenceNew();
/* Remove all the objects from ss AND DELETE THEM */
somf_TSortedSequence_somfDeleteAll(ss,ev);
_somFree (ss);
```

Original Class

somf_TCollection

Related Information

None.

somfFirst Method

Purpose

Gets the first object in a sorted sequence.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. somfFirst is a method name declared in multiple parents (for example: somf_TSequence, somf_TIterator.). You will probably have to fully qualify the name of the method (for example: somf_TSortedSequence_somfFirst). This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

```
seq->somfFirst(ev);
```

Syntax

somf_MCollectible **somfFirst** ();

Parameters

receiver A pointer to an object of class somf_TSortedSequence.

ev A pointer to the Environment structure for the calling method.

Restrictions and Limitations

None.

Returned Values

somf_MCollectible
A pointer to the first somf_MCollectible object in the sorted sequence.

SOMF_NIL Nothing is in the collection.

Example

```
somf_TSortedSequence ss;
Environment *ev;
somf_MOrderableCollectible obj;
ev = somGetGlobalEnvironment();
ss = somf_TSortedSequenceNew();
/* Determine the first object in ss */
obj = somf_TSortedSequence_somfFirst(ss,ev);
_somFree (ss);
```

Original Class

somf_TSequence (overridden here)

Related Information

somfAfter
somfBefore
somfLast

somfGetSequencingFunction Method

Purpose

Gets a pointer to the function used to compare objects in a sorted sequence, and consequently determines the sequence of the collection. This consequently reveals the sequence of the collection.

Syntax

```
somf_MBetterOrderableCompareFn
somfGetSequencingFunction ();
```

Parameters

receiver A pointer to an object of class somf_TSortedSequence.

ev A pointer to the Environment structure for the calling method.

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to the compare method used by this somf_TSortedSequence object.

Example

```
somf_TSortedSequence ss;
Environment *ev;
ev = somGetGlobalEnvironment();
ss = somf_TSortedSequenceNew();
if (_somfGetSequencingFunction(ss,ev) !=
     somf_MOrderableCollectibleClassData.somfCompare)
{
    somPrintf("\n What Compare Function are we using?\n");
}
_somFree (ss);
```

Original Class

somf_TSortedSequence

Related Information

somfSetSequencingFunction

somfLast Method

Purpose

Gets the last object in a sorted sequence.

C cannot handle methods from different classes having the same name when the inherit the name from different parents. somfLast is a method name declared in multiple parents (for example: somf_TSequenceIterator, somf_TSequence). You will probably have to fully qualify the name of the method (for example, somf_TSortedSequence_somfLast). This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

```
seq->somfLast(ev);
```

Syntax

somf_MCollectible **somfLast** ();

Parameters

receiver A pointer to an object of class somf_TSortedSequence.

ev A pointer to the Environment structure for the calling method.

Restrictions and Limitations

None.

Returned Values

somf_MCollectible
A pointer to the last somf_MCollectible object in the collection.

SOMF_NIL Nothing is in the collection.

Example

```
somf_TSortedSequence ss;
Environment *ev;
somf_MOrderableCollectible obj;
ev = somGetGlobalEnvironment();
ss = somf_TSortedSequenceNew();
/* Determine the last object in ss */
obj = somf_TSortedSequence_somfLast(ss,ev);
_somFree (ss);
```

Original Class

somf_TSequence

Related Information

somfAfter
somfBefore
somfFirst

somfMember Method

Purpose

Gets an object in a sorted sequence.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. If any child of somf_TCollection is used with somf_THashTable, then the name of the method will have to be fully qualified, so the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

```
d->somfMember(ev, obj);
```

Syntax

```
somf_MCollectible  somfMember(
    in somf_MCollectible obj);
```

Parameters

receiver A pointer to an object of class somf_TSortedSequence.

ev A pointer to the Environment structure for the calling method.

obj A pointer to the somf_MCollectible object that may be a member.

Restrictions and Limitations

None.

Returned Values

somf_MCollectible
A pointer to the object the method determined as a member.

SOMF_NIL The object was not found.

Example

```
somf_TSortedSequence ss;
<Your Class inheriting from somf_MOrderableCollectible> obj;
Environment *ev;
ev = somGetGlobalEnvironment();
ss = somf_TSortedSequenceNew();
obj =
   <Your Class inheriting from somf_MOrderableCollectible>New();
_somfAdd(ss, ev, obj);
if (_somfMember(ss, ev, obj) != SOMF_NIL)
    somPrintf("\n obj is a Member\n");
else
    somPrintf("\n ERROR: obj should be a Member\n");
_somFree (ss);
_somFree (obj);
```

Original Class

somf_TCollection

Related Information

somfOccurrencesOf

somfOccurrencesOf Method

Purpose

Determines the number of times an object is in a sorted sequence.

Syntax

```
long  somfOccurrencesOf(
    in somf_MCollectible obj);
```

Parameters

receiver A pointer to an object of class somf_TSortedSequence

ev A pointer to the Environment structure for the calling method.

obj A pointer to the somf_MCollectible to look for in the collection.

Restrictions and Limitations

None.

Returned Values

This method returns the number of times obj occurs in the sorted sequence.

Example

```
somf_TSortedSequence ss;
<Your Class which inherits from somf_MOrderableCollectible> obj;
Environment *ev;
ev = somGetGlobalEnvironment();
ss = somf_TSortedSequenceNew();
obj = <Your Class which inherits from
                              somf_MOrderableCollectible>New();
somPrintf("\n There are %d OccurrencesOf obj\n",
           _somfOccurrencesOf(ss, ev, obj));
_somFree (ss);
_somFree (obj);
```

Original Class

somf_TSequence

Related Information

somfMember

somfRemove Method

Purpose

Removes an object from a sorted sequence.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. somfRemove is a method name declared in multiple parents. You will probably have to fully qualify the method name. This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

```
d->somfRemove(ev, obj);
```

Syntax

```
somf_MCollectible  somfRemove(
    in somf_MCollectible obj);
```

Parameters

receiver A pointer to an object of class somf_TSortedSequence.

ev A pointer to the Environment structure for the calling method.

obj A pointer to the somf_MCollectible object to be removed from the collection.

Restrictions and Limitations

None.

Returned Values

There are two possible valid return values for this method:

somf_MCollectible
A pointer to the object that was actually removed.

SOMF_NIL The specified object was not found.

Example

```
somf_TSortedSequence ss;
<Your Class which inherits from somf_MOrderableCollectible> obj;
Environment *ev;
ev = somGetGlobalEnvironment();
ss = somf_TSortedSequenceNew();
obj = <Your Class which inherits from
          somf_MOrderableCollectible>New();
/* Remove obj from ss */
somf_TSortedSequence_somfRemove(ss,ev,obj);
_somFree (ss);
_somFree (obj);
```

Original Class

somf_TCollection

Related Information

somf_MCollectible

somfRemoveAll

somfRemoveAll Method

Purpose

Removes all of the objects from a sorted sequence.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Thus, if any child of somf_TCollection is used with somf_THashTable, then the name of the method will have to be fully qualified. This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

```
d->somfRemoveAll(ev);
```

Syntax

```
void  somfRemoveAll ();
```

Parameters

receiver A pointer to an object of class somf_TSortedSequence.

ev A pointer to the Environment structure for the calling method.

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_TSortedSequence ss;
Environment *ev;
ev = somGetGlobalEnvironment();
ss = somf_TSortedSequenceNew();
/* Remove all the objects from ss */
somf_TSortedSequence_somfRemoveAll(ss,ev);
_somFree (ss);
```

Original Class

somf_TCollection

Related Information

somfRemove

somfSetSequencingFunction Method

Purpose

Sets a pointer to the method used to compare objects in a sorted sequence, and consequently determines the sequence of the collection. This consequently determines the sequence of the collection.

Syntax

```
void  somfSetSequencingFunction(
    in somf_MBetterOrderableCompareFn
    fn);
```

Parameters

receiver A pointer to an object of class somf_TSortedSequence.

ev A pointer to the Environment structure for the calling method.

fn A pointer to the compare method that will be used by this somf_TSortedSequence object.

This should always be set to:

`somf_MOrderableCollectibleClassData.somfCompare.`

This specification is necessary because SOM needs a pointer to the original declaration of the method, which resides in somf_MOrderableCollectible Class. The somf_TSortedSequence object will use this pointer to access the somfCompare Method that was declared and defined in the object being inserted into, or removed from, the somf_TSortedSequence object.

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_TSortedSequence ss;
Environment *ev;
ev = somGetGlobalEnvironment();
ss = somf_TSortedSequenceNew();
_somfSetSequencingFunction(ss, ev,
         somf_MOrderableCollectibleClassData.somfCompare);
_somFree (ss);
```

Original Class

somf_TSortedSequence

Related Information

somfGetSequencingFunction

somfTSortedSequenceInitF Method

Purpose

Initializes a new sorted sequence, given the comparison method that it will use.

Note: You cannot override this method.

Syntax

```
somf_TSortedSequence
somfTSortedSequenceInitF(
    in somf_MBetterOrderableCompareFn
    testfn);
```

Parameters

receiver A pointer to an object of class somf_TSortedSequence.

ev A pointer to the Environment structure for the calling method.

testfn A pointer to the compare method that will be used by this somf_TSortedSequence object.

This should always be set to:

`somf_MOrderableCollectibleClassData.somfCompare.`

This specification is necessary because SOM needs a pointer to the original declaration of the method, which resides in somf _MOrderableCollectible Class. The somf_TSortedSequence object will use this pointer to access the somfCompare Method that was declared and defined in the object being inserted into, or removed from, the somf_TSortedSequence object.

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to an initialized somf_TSortedSequence object.

Example

```
somf_TSortedSequence s1;
Environment *ev;
ev = somGetGlobalEnvironment();
s1 = somf_TSortedSequenceNew();
_somfTSortedSequenceInitF(s1, ev,
         somf_MOrderableCollectibleClassData.somfCompare);
_somFree (s1);
```

Original Class

somf_TSortedSequence

Related Information

somfTSortedSequenceInitS

somfTSortedSequenceInitS Method

Purpose

Initializes a new sorted sequence, setting it equal to another given sorted sequence. The method sets the new sorted sequence equal to a specified source sorted sequence.

Note: You cannot override this method.

Syntax

```
somf_TSortedSequence
somfTSortedSequenceInitS(
    in somf_TSortedSequences);
```

Parameters

receiver A pointer to an object of class somf_TSortedSequence.

ev A pointer to the Environment structure for the calling method.

s A pointer to the somf_TSortedSequence object to which the new sorted sequence will be equal.

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to an initialized somf_TSortedSequence object.

Example

```
somf_TSortedSequence s1;
somf_TSortedSequence s2;
Environment *ev;
ev = somGetGlobalEnvironment();
s1 = somf_TSortedSequenceNew();
s2 = somf_TSortedSequenceNew();
_somfTSortedSequenceInitS(s2, ev, s1);
_somFree (s1);
_somFree (s2);
```

Original Class

somf_TSortedSequence

Related Information

somfTSortedSequenceInitF

Chapter 24. somf_TSortedSequenceIterator Class

This class defines an iterator for the somf_TSortedSequence Class that will iterate over all of the objects in a sorted sequence.

Member Name:

tssitr

Import Name:

GOSSOMUC

Base Class:

somf_TSequenceIterator

Metaclass:

SOMClass

Ancestor Classes:

somf_TSequenceIterator
somf_TIterator
SOMObject

Subclasses:

None.

Types:

None.

Attributes:

None.

New Methods:

somfStartHere
somfTSortedSequenceIteratorInit

Overriding Methods:

somfFirst
somfLast
somfNext
somfPrevious
somfRemove

somfFirst Method

Purpose

Resets the iterator and returns the first object of a sorted sequence. The method returns the first object of the somf_TSortedSequence collection that corresponds to the specified iterator.

This method resets the iterator to the beginning of the sorted sequence even when other operations on the collection cause the iterator to be invalidated. In the second case, this method revalidates the iterator.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. somfFirst is a method name declared in multiple parents. You will probably have to fully qualify the name of the method. This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

```
itr->somfFirst(ev);
```

Syntax

somf_MCollectible **somfFirst** ();

Parameters

receiver A pointer to an object of class somf_TSortedSequenceIterator.

ev A pointer to the Environment structure for the calling method.

Restrictions and Limitations

None.

Returned Values

somf_MCollectible
A pointer to the first somf_MCollectible object in the sorted sequence

SOMF_NIL Is returned if the collection is empty.

Example

```
somf_TSortedSequence ss;
Environment *ev;
somf_TSortedSequenceIterator itr;
<Your class inheriting from somf_MOrderableCollectible> itrobj;
ev = somGetGlobalEnvironment();
ss = somf_TSortedSequenceNew();
itr = somf_TSortedSequenceIteratorNew();
_somfTSortedSequenceIteratorInit(itr, ev, ss);
/* Iterate through the TSortedSequence */
itrobj =
   (<Your class that inherits from somf_MOrderableCollectible>*)
    somf_TSortedSequenceIterator_somfFirst(itr,ev);
while (itrobj != SOMF_NIL)
/* Do something with itrobj */
{itrobj =
       (<Your class inheriting from somf_MOrderableCollectible>*)
        _somfNext(itr,ev);}
_somFree (ss);
_somFree (itr);
```

Original Class

somf_TIterator

Related Information

somfNext
somfStartHere

somfLast Method

Purpose

Gets the last object in a sorted sequence.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. somfLast is a method name declared in multiple parents (for example: somf_TSequenceIterator, somf_TSequence). You will probably have to fully qualify the name of the method. This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

```
itr->somfLast(ev);
```

Syntax

somf_MCollectible **somfLast** ();

Parameters

receiver A pointer to an object of class somf_TSortedSequenceIterator.

ev A pointer to the Environment structure for the calling method.

Restrictions and Limitations

None.

Returned Values

somf_MCollectible
A pointer to the last somf_MCollectible object in the sorted sequence.

SOMF_NIL Is returned if the collection is empty.

Example

```
somf_TSortedSequence ss;
Environment *ev;
somf_TSortedSequenceIterator itr;
<Your class inheriting from somf_MOrderableCollectible> itrobj;
ev = somGetGlobalEnvironment();
ss = somf_TSortedSequenceNew();
itr = somf_TSortedSequenceIteratorNew();
_somfTSortedSequenceIteratorInit(itr, ev, ss);
/* Go to the last object in ss */
itrobj =
   (<Your class which inherits from somf_MOrderableCollectible>*)
    somf_TSortedSequenceIterator_somfLast(itr,ev);
_somFree (ss);
_somFree (itr);
```

Original Class

somf_TSequenceIterator (overridden here)

Related Information

somfPrevious

somfNext Method

Purpose

Gets the next object in a sorted sequence. Objects are retrieved in an order that reflects the "ordered-ness" of the sorted sequence (or the lack of ordering on the sorted sequence objects).

If the somf_TSortedSequence collection has changed since the last time the somfFirst method was called, the iterator becomes invalid and will fail if asked to find the next object. For example, if the collection's somfAdd method were called after starting to iterate through the collection, the iterator then would not allow iteration to continue. The iterator must be reset, and the easiest way to do that is to call the iterator's somfFirst method and start over.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. Thus, if any child of somf_TIterator is used with a child of somf_TPrimitiveLinkedListIterator, then the name of the method will have to be fully qualified. This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

```
itr->somfNext(ev);
```

Syntax

somf_MCollectible **somfNext** ();

Parameters

receiver A pointer to an object of class somf_TSortedSequenceIterator.

ev A pointer to the Environment structure for the calling method.

Restrictions and Limitations

None.

Returned Values

somf_MCollectible
A pointer to the next somf_MCollectible object in the sorted sequence.

SOMF_NIL The end of the collection has been reached.

Example

```
somf_TSortedSequence ss;
Environment *ev;
somf_TSortedSequenceIterator itr;
<Your class inheriting from somf_MOrderableCollectible> itrobj;
ev = somGetGlobalEnvironment();
ss = somf_TSortedSequenceNew();
itr = somf_TSortedSequenceIteratorNew();
_somfTSortedSequenceIteratorInit(itr, ev, ss);
/* Iterate through the TSortedSequence */
itrobj =
   (<Your class inheriting from somf_MOrderableCollectible>*)
    somf_TSortedSequenceIterator_somfFirst(itr,ev);
while (itrobj != SOMF_NIL)
{
  /* Do something with itrobj */
    itrobj =
    (<Your class inheriting from somf_MOrderableCollectible>*)
        _somfNext(itr,ev);
}
_somFree (ss);
_somFree (itr);
```

Original Class

somf_TIterator

Related Information

somfFirst
somfStartHere

somfPrevious Method

Purpose

Gets the previous object in a sorted sequence.

If the somf_TSortedSequence collection has changed since somfLast was called, the iterator becomes invalid and will fail if asked to find the previous object. If the collection's somfAdd method were called after starting to iterate through the collection, the iterator then would not allow iteration to continue. The iterator must be reset. The easiest way is to call the iterator's somfLast method and restart.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. If any child of somf_TSequenceIterator is used with somf_TPrimitiveLinkedListIterator, the name of the method will have to be fully qualified, so the linker can tell them apart. This is not a problem in C++. In C++ you can referenced this method as:

```
itr->somfPrevious(ev);
```

Syntax

somf_MCollectible **somfPrevious** ();

Parameters

receiver A pointer to an object of class somf_TSortedSequenceIterator.

ev A pointer to the Environment structure for the calling method.

Restrictions and Limitations

None.

Returned Values

somf_MCollectible
A pointer to the previous object in the sorted sequence collection.

SOMF_NIL The beginning of the collection has been reached.

Example

```
somf_TSortedSequence ss;
Environment *ev;
somf_TSortedSequenceIterator itr;
<Your class inheriting from somf_MOrderableCollectible> itrobj;
ev = somGetGlobalEnvironment();
ss = somf_TSortedSequenceNew();
itr = somf_TSortedSequenceIteratorNew();
_somfTSortedSequenceIteratorInit(itr, ev, ss);
/* Go to the next to the last object in ss */
somf_TSortedSequenceIterator_somfLast(itr,ev);
itrobj =
   (<Your class which inherits from somf_MOrderableCollectible>*)
    _somfPrevious(itr,ev);
_somFree (ss);
_somFree (itr);
```

Original Class

somf_TSortedSequenceIterator (overridden here)

Related Information

somfLast

somfRemove Method

Purpose

Removes the current object just returned by a previous method from a sorted sequence.

somfRemove is the only way to remove an object from a sorted sequence during iteration. However, if multiple iterators are in process, all other iterators are invalidated, just as if some other kind of change had occurred in the sorted sequence.

If the collection has changed (other than through the use of somfRemove of this iterator) since the last time somfFirst or somfLast was called, this method will fail.

C cannot handle methods from different classes having the same name when they inherit the name from different parents. somfRemove is a method name declared in multiple parents. You will probably have to fully qualify the method name. This is the only way the linker can tell them apart. This is not a problem in C++. In C++ you could have referenced this method as:

```
itr->somfRemove(ev);
```

Syntax

```
void  somfRemove ();
```

Parameters

receiver A pointer to an object of class somf_TSortedSequenceIterator.

ev A pointer to the Environment structure for the calling method.

Restrictions and Limitations

None.

Returned Values

None.

Example

```
somf_TSortedSequence ss;
Environment *ev;
somf_TSortedSequenceIterator itr;
<Your class inheriting from somf_MOrderableCollectible> itrobj;
ev = somGetGlobalEnvironment();
ss = somf_TSortedSequenceNew();
itr = somf_TSortedSequenceIteratorNew();
_somfTSortedSequenceIteratorInit(itr, ev, ss);
/* Use the Iterator's Remove to remove the next to last object */
itrobj =
   (<Your class which inherits from somf_MOrderableCollectible>*)
    somf_TSortedSequenceIterator_somfLast(itr,ev);
itrobj =
   (<Your class which inherits from somf_MOrderableCollectible>*)
    _somfPrevious(itr,ev);
somf_TSortedSequenceIterator_somfRemove(itr,ev);
_somFree (ss);
_somFree (itr);
```

Original Class

somf_TIterator

Related Information

None.

somfStartHere Method

Purpose

Begins Iterating through a somf_TSortedSequence Class, starting at a given object. Iteration begins at the given object.

Syntax

```
somf_MOrderableCollectible
somfStartHere(
    in
    somf_MOrderableCollectible obj);
```

Parameters

receiver A pointer to an object of class somf_TSortedSequenceIterator.

ev A pointer to the Environment structure for the calling method.

obj A pointer to the somf_MOrderableCollectible object where iteration will begin.

Restrictions and Limitations

None.

Returned Values

somf_MCollectible
A pointer to the somf_MCollectible object where iteration started

SOMF_NIL Is returned if the collection is empty.

Example

```
somf_TSortedSequence ss;
Environment *ev;
somf_TSortedSequenceIterator itr;
<Your class inheriting from somf_MOrderableCollectible> obj;
<Your class inheriting from somf_MOrderableCollectible> itrobj;
ev = somGetGlobalEnvironment();
ss = somf_TSortedSequenceNew();
obj = <Your Class that inherits from \n
somf_MOrderableCollectible>New();
itr = somf_TSortedSequenceIteratorNew();
_somfTSortedSequenceIteratorInit(itr, ev, ss);
/* Iterate through the TSortedSequence starting at obj */
itrobj =
   (<Your class which inherits from somf_MOrderableCollectible>*)
    _somfStartHere(itr,ev,obj);
while (itrobj != SOMF_NIL)
{
    /* Do something with itrobj */
    itrobj =
      (<Your class inheriting from somf_MOrderableCollectible>*)
       _somfNext(itr,ev);
}
_somFree (ss);
_somFree (itr);
```

Original Class

somf_TSortedSequenceIterator

Related Information

somfFirst
somfNext

somfTSortedSequenceIteratorInit Method

Purpose

Initializes a new somf_TSortedSequenceIterator object, given the collection of class somf_TSortedSequence Class over which it will iterate.

This is one of three ways to initialize a somf_TSortedSequenceIterator to point to an instance of a somf_TSortedSequence. The other two ways are:

- to use the somf_TSortedSequence class's somfCreateSequenceIterator method.
- to use somf_TSortedSequence's somfCreateIterator Method.

Note: You cannot override this method.

Syntax

```
somf_TSortedSequenceIterator
somfTSortedSequenceIteratorInit(
    in somf_TSortedSequence h);
```

Parameters

receiver A pointer to an object of class somf_TSortedSequenceIterator.

ev A pointer to the Environment structure for the calling method.

h A pointer to the sorted sequence that the receiving object will iterate over.

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to an initialized somf_TSortedSequenceIterator object.

Example

```
somf_TSortedSequence ss;
Environment *ev;
somf_TSortedSequenceIterator itr;
ev = somGetGlobalEnvironment();
ss = somf_TSortedSequenceNew();
itr = somf_TSortedSequenceIteratorNew();
_somfTSortedSequenceIteratorInit(itr, ev, ss);
_somFree (ss);
_somFree (itr);
```

Original Class

somf_TSortedSequenceIterator

Related Information

None.

Chapter 25. somf_TSortedSequenceNode Class

The somf_TSortedSequenceNode class defines a node in a tree. Objects inserted into a node must be of the somf_MOrderableCollectible Class. Each node contains a key and links to a left child and a right child. An object of class somf_TSortedSequenceNode is used by the somf_TSortedSequence Class

somf_TSortedSequenceNode class and methods are for creating a new class that:

- needs linkable nodes between objects of the class
- inherits from somf_TSortedSequence, and it would be appropriate to override some methods of the somf_TSortedSequence class to define additional functionality for those methods.

This class is not thread-safe.

This class is reentrant.

Member Name:

tssnode

Import Name:

GOSSOMUC

Base Class:

SOMObject

Metaclass:

SOMClass

Ancestor Classes:

SOMObject

Subclasses:

None.

Types:

None.

Attributes:

None.

New Methods:

somfGetLeftChild
somfGetRightChild
somfGetParent

somfGetKey
somfGetRed
somfSetParent
somfSetLeftChild
somfSetRightChild
somfSetKey
somfSetRed
somfSetRedOn
somfTSortedSequenceNodeInitTMT
somfTSortedSequenceNodeInitTM
somfTSortedSequenceNodeInitT

Overriding Methods:

somDefaultInit

somfGetKey Method

Purpose

Gets the key to a node.

Note: You cannot override this method.

Syntax

```
somf_MOrderableCollectible
somfGetKey ();
```

Parameters

receiver A pointer to an object of class somf_TSortedSequenceNode.

ev A pointer to the Environment structure for the calling method.

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to the somf_MOrderableCollectible key.

Original Class

somf_TSortedSequenceNode

Related Information

somfSetKey

somfGetLeftChild Method

Purpose

Gets the left child of a node.

Note: You cannot override this method.

Syntax

```
somf_TSortedSequenceNode
somfGetLeftChild();
```

Parameters

receiver A pointer to an object of class somf_TSortedSequenceNode.

ev A pointer to the Environment structure for the calling method.

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to the left child of the node.

Original Class

somf_TSortedSequenceNode

Related Information

somfSetLeftChild

somfGetParent Method

Purpose

Gets the parent of a node.

Note: You cannot override this method.

Syntax

```
somf_TSortedSequenceNode
somfGetParent ();
```

Parameters

receiver A pointer to an object of class somf_TSortedSequenceNode.

ev A pointer to the Environment structure for the calling method.

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to the parent of the node.

Original Class

somf_TSortedSequenceNode

Related Information

somfSetParent

somfGetRed Method

Purpose

Determines whether a node is red or black.

Note: For a discussion of Red-Black Trees, you can refer to *Algorithms in C++* by Robert Sedgewick (Addison-Wesley Publishing Company, 1992).

Note: You cannot override this method.

Syntax

```
boolean somfGetRed ();
```

Parameters

receiver A pointer to an object of class somf_TSortedSequenceNode.

ev A pointer to the Environment structure for the calling method.

on One of these two choices:

- **TRUE** The node is red.
- **FALSE** The node is black.

Restrictions and Limitations

None.

Returned Values

None.

Original Class

somf_TSortedSequenceNode

Related Information

somfSetRed
somfSetRedOn

somfGetRightChild Method

Purpose

Gets the right child of a node.

Note: You cannot override this method.

Syntax

```
somf_TSortedSequenceNode
somfGetRightChild ();
```

Parameters

receiver A pointer to an object of class somf_TSortedSequenceNode.

ev A pointer to the Environment structure for the calling method.

Restrictions and Limitations

None.

Returned Values

This method returns the pointer to the right child of the node.

Original Class

somf_TSortedSequenceNode

Related Information

somfSetRightChild

somfSetKey Method

Purpose

Sets the key to a node, given a pointer to a key object.

Note: You cannot override this method.

Syntax

```
void somfSetKey(
    in somf_MOrderableCollectible k);
```

Parameters

receiver A pointer to an object of class somf_TSortedSequenceNode.

ev A pointer to the Environment structure for the calling method.

k A pointer to the somf_MOrderableCollectible key.

Restrictions and Limitations

None.

Returned Values

None.

Example

None.

Original Class

somf_TSortedSequenceNode

Related Information

somfGetKey

somfSetLeftChild Method

Purpose

Sets the left child of a node, given a pointer to an object that will be the left child.

Note: You cannot override this method.

Syntax

```
void somfSetLeftChild(
    in somf_TSortedSequenceNode n);
```

Parameters

receiver A pointer to an object of class somf_TSortedSequenceNode.

ev A pointer to the Environment structure for the calling method.

n A pointer to the left child of the node.

Restrictions and Limitations

None.

Returned Values

None.

Example

None.

Original Class

somf_TSortedSequenceNode

Related Information

somfGetLeftChild

somfSetParent Method

Purpose

Sets the parent of a node, given an object to be used as the parent node.

Note: You cannot override this method.

Syntax

```
void somfSetParent(
    in somf_TSortedSequenceNode n);
```

Parameters

receiver A pointer to an object of class somf_TSortedSequenceNode.

ev A pointer to the Environment structure for the calling method.

n A pointer to the parent node of the receiving-object node.

Restrictions and Limitations

None.

Returned Values

None.

Example

None.

Original Class

somf_TSortedSequenceNode

Related Information

somfGetParent

somfSetRed Method

Purpose

Sets a node to red or black.

Note: For a discussion of Red-Black Trees, you can refer to *Algorithms in C++* by Robert Sedgewick (Addison-Wesley Publishing Company, 1992).

Note: You cannot override this method.

Syntax

```
void somfSetRed (in boolean on);
```

Parameters

receiver A pointer to an object of class somf_TSortedSequenceNode

ev A pointer to the Environment structure for the calling method.

on One of these two choices:

- **TRUE** The node is red.
- **FALSE** The node is black.

Restrictions and Limitations

None.

Returned Values

None.

Example

None.

Original Class

somf_TSortedSequenceNode

Related Information

somfSetRedOn
somfGetRed

somfSetRedOn Method

Purpose

Sets a node to red.

Note: For a discussion of Red-Black Trees, you can refer to *Algorithms in C++* by Robert Sedgewick (Addison-Wesley Publishing Company, 1992).

Note: You cannot override this method.

Syntax

```
void somfSetRedOn ();
```

Parameters

receiver A pointer to an object of class somf_TSortedSequenceNode.

ev A pointer to the Environment structure for the calling method.

Restrictions and Limitations

None.

Returned Values

None.

Example

None.

Original Class

somf_TSortedSequenceNode

Related Information

somfSetRed
somfGetRed

somfSetRightChild Method

Purpose

Sets the right child of a node, given a pointer to an object that will be the right child.

Note: You cannot override this method.

Syntax

```
void somfSetRightChild(
    in somf_TSortedSequenceNode n);
```

Parameters

receiver A pointer to an object of class somf_TSortedSequenceNode.

ev A pointer to the Environment structure for the calling method.

n A pointer to the right child of the node.

Restrictions and Limitations

None.

Returned Values

None.

Example

None.

Original Class

somf_TSortedSequenceNode

Related Information

somfGetRightChild

somfTSortedSequenceNodeInitT Method

Purpose

Initializes a new somf_TSortedSequenceNode node, given its left child. The method call also specifies a somf_TSortedSequenceNode object to be used as the left child of the new node.

Note: You cannot override this method.

Syntax

```
somf_TSortedSequenceNode
somfTSortedSequenceNodeInitT
    in somf_TSortedSequenceNode n1);
```

Parameters

receiver A pointer to an object of class somf_TSortedSequenceNode.

ev A pointer to the Environment structure for the calling method.

n1 A pointer to the left child of the new somf_TSortedSequenceNode object.

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to an initialized somf_TSortedSequenceNode object.

Example

None.

Original Class

somf_TSortedSequenceNode

Related Information

somfTSortedSequenceNodeInitTMT
somfTSortedSequenceNodeInitTM

somfTSortedSequenceNodeInitTM Method

Purpose

Initializes a new somf_TSortedSequenceNode node, given its left child and a key to the new node. The method call also specifies a somf_TSortedSeq uenceNode object to be used as the left child of the new node, and a somf_MOrderableCollectible object to be used as the key to the new node.

Note: You cannot override this method.

Syntax

```
somf_TSortedSequenceNode
somfTSortedSequenceNodeInitTM
    in somf_TSortedSequenceNode n1,
    in
    somf_MOrderableCollectible obj);
```

Parameters

receiver A pointer to an object of class somf_TSortedSequenceNode.

ev A pointer to the Environment structure for the calling method.

n1 A pointer to the left child of the new somf_TSortedSequenceNode object.

obj A pointer to the key of the new somf_TSortedSequenceNode object.

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to an initialized somf_TSortedSequenceNode object.

Example

None.

Original Class

somf_TSortedSequenceNode

Related Information

somfTSortedSequenceNodeInitTMT
somfTSortedSequenceNodeInitT

somfTSortedSequenceNodeInitTMT Method

Purpose

Initializes a new somf_TSortedSequenceNode node, given a key to the new node and its left and right children. The method call specifies a somf_MOrderableCollectible object to be used as the key to the new node, and two somf_TSortedSequenceNode objects to be used as the left and right children of the new node.

Note: You cannot override this method.

Syntax

```
somf_TSortedSequenceNode
somfTSortedSequenceNodeInitTMT
    in somf_TSortedSequenceNode n1,
    in somf_MOrderableCollectible
    obj,
    in somf_TSortedSequenceNode n2);
```

Parameters

receiver A pointer to an object of class somf_TSortedSequenceNode.

ev A pointer to the Environment structure for the calling method.

n1 A pointer to the left child of the new somf_TSortedSequenceNode object.

obj A pointer to the key of the new somf_TSortedSequenceNode object.

n2 A pointer to the right child of the new somf_TSortedSequenceNode object.

Restrictions and Limitations

None.

Returned Values

This method returns a pointer to an initialized somf_TSortedSequenceNode object.

Example

None.

Original Class

somf_TSortedSequenceNode

Related Information

somfTSortedSequenceNodeInitTM
somfTSortedSequenceNodeInitT

Index

Communicating Your Comments to IBM

OS/390
SOMobjects
Programmer's Reference,
Volume 3

Publication No. SC28-1999-01

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply.

Feel free to comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. However, the comments you send should pertain to only the information in this manual and the way in which the information is presented. To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

If you are mailing an RCF from a country other than the United States, you can give the RCF to the local IBM branch office or IBM representative for postage-paid mailing.

- If you prefer to send comments by mail, use the RCF at the back of this book.
- If you prefer to send comments by FAX, use this number:
  - FAX: (International Access Code)+1+914+432-9405
- If you prefer to send comments electronically, use this network ID:
  - IBMLink: (United States customers only): KGNVMC(MHVRCFS)
  - IBM Mail Exchange: USIB6TC9 at IBMMAIL
  - Internet e-mail: mhvrcfs@us.ibm.com
  - World Wide Web: http://www.s390.ibm.com/os390

Make sure to include the following in your note:

- Title and publication number of this book
- Page number or topic to which your comment applies

Optionally, if you include your telephone number, we will be able to respond to your comments by phone.

Reader's Comments — We'd Like to Hear from You

OS/390
SOMobjects
Programmer's Reference,
Volume 3

Publication No. SC28-1999-01

You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you. Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Today's date: _____

What is your occupation?

Newsletter number of latest Technical Newsletter (if any) concerning this publication:

How did you use this publication?

[ ]	As an introduction	[ ]	As a text (student)
[ ]	As a reference manual	[ ]	As a text (instructor)
[ ]	For another purpose (explain)		

_____

_____

Is there anything you especially like or dislike about the organization, presentation, or writing in this manual? Helpful comments include general usefulness of the book; possible additions, deletions, and clarifications; specific errors and omissions.

Page Number: Comment:

_____ _____
Name Address

_____ _____
Company or Organization

_____ _____
Phone No.

Reader's Comments — We'd Like to Hear from You
SC28-1999-01

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Department 55JA, Mail Station P384
522 South Road
Poughkeepsie NY 12601-5400

Fold and Tape | **Please do not staple** | Fold and Tape

Cut or Fold
Along Line

File Number: S370/S390-79
Program Number: 5647-A01

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

Drop in
Back Cover
Image Here.

SC28-1999-01