Overview

The Object class is the parent class for all other classes in SAS/AF software. As the parent class, Object contains the attributes, methods, events, and event handlers that are inherited by all other classes, including the visual controls, non-visual models, and legacy objects supplied with SAS/AF software.

Class:

sashelp.fsp.Object.class

Using the Object Class

You can use the Object class as the parent class for many of the classes that you create with SAS/AF software. The Object class contains the base information that you need to create a class and enable communication with other objects. By subclassing the Object class and combining it with the SAS Component Object Model (SCOM) architecture, you can build components using a flexible application framework that offers improved component communication.

Instance Variables

Instance variables contain data associated with an object, such as its description, the types of messages it can receive, the objects it can receive messages from, and the objects it can send messages to. The information contained in the instance variables is used by some of the Object class methods (for example, the _broadcast method) to broadcast messages between objects.

Every object can have the following instance variables.
_receivers_
contains a list of objects that receive broadcast messages from the object. When an object broadcasts a message through the _broadcast method, the message is sent to all objects in this _receivers list. Thus, this list enables an object to send messages to many other objects with one method call.

If the sending and receiving objects exist in the same frame when you build an application, you can create an object's _receivers_ list in the development environment using the Object Links window. One advantage of this is that because the objects must already exist, you know the names of the objects you want to put on or remove from the _receivers list. Therefore, you can use the object names rather than object identifiers to set up the communications link.

By contrast, if you have any objects that do not exist when the application is built but that need to send and receive messages with the _broadcast method, you can use the _addReceiver method to add these objects to a _receivers_ list. See _addReceiver in this class. To remove objects from a _receivers_ list, you can use the _deleteReceiver method described in this class.

_senders_
contains a list of the objects that can send a message to the object. In other words, _senders_ contains a list of objects from which an object can receive messages. When an object is added to another object's _receivers list through the _addReceivers method, the other object is automatically added to its _senders_ list. For example, OBJ ECT A is automatically added to OBJ ECT B's _senders list when OBJ ECT B is added to OBJ ECT A's _receivers list. If OBJ ECT B does not have a _senders_ list, it is created automatically. The _senders_ list is not updated when receivers are added through the Object Links window.

_received_
contains a list of message names that identify the broadcast message strings an object can receive through a _broadcast method. For example, if you have a Text Entry object that should receive only _setColorName and _setAttr messages, you would specify these messages in the Text Entry's _receive_ list. You can add messages to the Object class' _receive_ list by editing the object's CLASS entry and selecting _Receive list from the Additional Attributes list.

The _receive list acts as a filter for messages broadcast from other objects. Only messages that are the names of items in the _receive_ list are delivered. (The value of each item in the _receive_ list is ignored.) For example, if you edit the _receive_ list for a class and specify that the class should receive only messages _changed and _deleted, the _receive_list looks like this:

```
_received=(_changed=' ' _deleted=' ')
```

If an object does not have a _receive_ list (that is, there is no item named _receive_), the software checks the _receive list of the class from which that object was created. All objects of a particular class respond to broadcast messages with the same _receive method. If the _receive_ list exists but is empty (or is not a valid list), broadcast messages are not delivered.

---

**Object Class Attributes**

Public attributes specified for the Object Class class are described here.

attachedInterface"Object Class: attachedInterface" on page 91
Object Class: CBTFrameName

Returns the CBT frame that is displayed when a CBT entry is assigned as the help for the object
Type: Character
Valid Values:

Object Class: contentsUpdatedAttributes

Returns or sets the attributes which send a 'contents updated' event when their value is changed
Type: Character
Valid Values: \sashelp.classes.defaultattributevalues.scl
Editor: sashelp.classes.selectAttributesEditor.frame
Object Class: defaultAttribute

Returns or sets the attribute name used for the object's default attribute

Type: Character
Valid Values: \sashelp.classes.defaultattributevalues.scl

Object Class: description

Returns the short description for the object

Type: Character
Valid Values:

Object Class: errorMessage

Returns or sets the appropriate error message

Type: Character
Valid Values:

Object Class: frameID

Returns the ID of the frame that contains the object

Type: Object
Valid Values:
Object Class: help

Returns or sets the help topic that appears when the user selects object help (such as What's This? help)

Type: Character
Valid Values:
Editor: sashelp.classes.objectHelpEditor.frame

Object Class: model

Returns or sets the name of the object that serves as the model (or server) in the model-viewer relationship with this object

Type: Character
Valid Values: \sashelp.classes.modelvalues.scl

Object Class: name

Returns or sets the name of the object

Type: Character
Valid Values:

Object Class: ownerID

Returns the ID of the owner object if applicable

Type: Object
Valid Values:
Object Class Methods

Methods specified for the Object Class class are described here.

_addAttribute("Object Class: _addAttribute" on page 95
_addEvent("Object Class: _addEvent" on page 95
_addEventHandler("Object Class: _addEventHandler" on page 96
_addLink("Object Class: _addLink" on page 97
_addMethod("Object Class: _addMethod" on page 98
_addReceiver("Object Class: _addReceiver" on page 99
_attributesDialog("Object Class: _attributesDialog" on page 99
_binit("Object Class: _binit" on page 99
_broadcast("Object Class: _broadcast" on page 100
_bterm("Object Class: _bterm" on page 100
_changeAttribute("Object Class: _changeAttribute" on page 101
_changeEvent("Object Class: _changeEvent" on page 101
_changeEventHandler("Object Class: _changeEventHandler" on page 102
_changeMethod("Object Class: _changeMethod" on page 103
_deleteAttribute("Object Class: _deleteAttribute" on page 104
_deleteEvent("Object Class: _deleteEvent" on page 105
_deleteEventHandler("Object Class: _deleteEventHandler" on page 105
_deleteInstanceMethod("Object Class: _deleteInstanceMethod" on page 105
_deleteLink("Object Class: _deleteLink" on page 106
_deleteMethod("Object Class: _deleteMethod" on page 107
_deleteReceiver("Object Class: _deleteReceiver" on page 108
_getAttribute("Object Class: _getAttribute" on page 108
_getAttributes("Object Class: _getAttributes" on page 109
_getAttributeValue("Object Class: _getAttributeValue" on page 110
_getClass("Object Class: _getClass" on page 110
_getData("Object Class: _getData" on page 111
_getEvent("Object Class: _getEvent" on page 111
_getEventHandler("Object Class: _getEventHandler" on page 112
_getEventHandlers("Object Class: _getEventHandlers" on page 112
_getEvents("Object Class: _getEvents" on page 113
_getMethodList("Object Class: _getMethodList" on page 114
_getLink("Object Class: _getLink" on page 114
_getLinks("Object Class: _getLinks" on page 115
_getMethod("Object Class: _getMethod" on page 115
_getMethodD("Object Class: _getMethodD" on page 117
_getMethods("Object Class: _getMethods" on page 117
_getModelID("Object Class: _getModelID" on page 118
_getProperties("Object Class: _getProperties" on page 119
_getSignatureString("Object Class: _getSignatureString" on page 119
_hasMethod("Object Class: _hasMethod" on page 119
_help("Object Class: _help" on page 120
_init("Object Class: _init" on page 120
_initConstructor("Object Class: _initConstructor" on page 121
_isA("Object Class: _isA" on page 121
_new("Object Class: _new" on page 122
_newClass("Object Class: _newClass" on page 122
_onAttributeChange("Object Class: _onAttributeChange" on page 123
_onContentsUpdated("Object Class: _onContentsUpdated" on page 123
_printObject("Object Class: _printObject" on page 123
_purgeEventHandler("Object Class: _purgeEventHandler" on page 124
Object Class: _addAttribute

Adds an attribute to a class or object

Syntax

return = objectName_addAttribute( metadataList );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>Numeric</td>
<td>Return</td>
<td>returns a value indicating whether the attribute was successfully added: 0 if successful, 1 if not successful</td>
</tr>
<tr>
<td>metadataList</td>
<td>List</td>
<td>Update</td>
<td>specifies a list containing the named items for attribute metadata</td>
</tr>
</tbody>
</table>

Details

The _addAttribute method is valid on both class and instance.

Object Class: _addEvent

Registers an event on a class or object

Syntax

return = objectName_addEvent( metadataList );
return = objectName_addEvent( eventName );
return = objectName_addEvent( eventName, executeType );
return = objectName_addEvent( eventName, executeType, method );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>Numeric</td>
<td>Return</td>
<td>returns a value indicating whether the event was successfully added: 0 if successful, 1 if not successful</td>
</tr>
<tr>
<td>metadataList</td>
<td>List</td>
<td>Update</td>
<td>specifies a list containing the named items for the event metadata</td>
</tr>
<tr>
<td>eventName</td>
<td>Character</td>
<td>Input</td>
<td>specifies the name of the event to add</td>
</tr>
<tr>
<td>executeType</td>
<td>Character</td>
<td>Input</td>
<td>specifies when the event gets executed. BEFORE or AFTER indicates the event is automatically sent when the specified method executes. MANUAL implies the event is sent programmatically in SCL using t</td>
</tr>
<tr>
<td>method</td>
<td>Character</td>
<td>Input</td>
<td>specifies the method that triggers the event; used only if executeType is BEFORE or AFTER</td>
</tr>
</tbody>
</table>

**Details**

The _addEvent signature that passes metadataList is valid on both class and instance; the remaining signatures are valid on instance only.

---

**Object Class: _addEventHandler**

Specifies the action that takes place when a particular even occurs

---

**Syntax**

\[
\text{return} = \text{objectName}_{-}\text{addEventHandler}(\text{metadataList});
\]

\[
\text{return} = \text{objectName}_{-}\text{addEventHandler}(\text{senderName, event, action});
\]

\[
\text{return} = \text{objectName}_{-}\text{addEventHandler}(\text{senderID, event, action});
\]
Object Class: _addLink

Adds an attribute link on an object

Syntax

\[
\text{return} = \text{objectName\_addLink( attributeToLink, sourceObject, sourceAttribute );}
\]

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>Numeric</td>
<td>Return</td>
<td>returns a value indicating whether the link was successfully added: 0 if successful, 1 if not successful</td>
</tr>
<tr>
<td>attributeToLink</td>
<td>Character</td>
<td>Input</td>
<td>specifies the name of the attribute on the instance to which you want to link</td>
</tr>
<tr>
<td>sourceObject</td>
<td>Object</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sourceAttribute</td>
<td>Character</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Object Class: _addMethod

Adds a method to a class or object

Syntax

```java
return = objectName_addMethod( metadataList, returnMethodID );
return = objectName_addMethod( metadataList );
return = objectName_addMethod( methodName, entry, label );
return = objectName_addMethod( methodName, entry, label, signature );
return = objectName_addMethod( methodName, entry, label, signature, returnMethodID );
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sourceObject</td>
<td>Object</td>
<td>Input</td>
<td>specifies the object you link from</td>
</tr>
<tr>
<td>sourceAttribute</td>
<td>Character</td>
<td>Input</td>
<td>specifies the attribute on the sourceObject to link</td>
</tr>
<tr>
<td>Return</td>
<td>Numeric</td>
<td>Return</td>
<td>returns a value indicating whether the method was successfully added: 0 if successful, 1 if not successful</td>
</tr>
<tr>
<td>metadataList</td>
<td>List</td>
<td>Input</td>
<td>specifies a list of named items containing the method metadata</td>
</tr>
<tr>
<td>returnMethodID</td>
<td>Numeric</td>
<td>Output</td>
<td>returns the unique number assigned by the system to identify a specific method</td>
</tr>
<tr>
<td>methodName</td>
<td>Character</td>
<td>Input</td>
<td>specifies the name of the method to add</td>
</tr>
<tr>
<td>entry</td>
<td>Character</td>
<td>Input</td>
<td>specifies the catalog entry containing the implementation for the method</td>
</tr>
<tr>
<td>label</td>
<td>Character</td>
<td>Input</td>
<td>specifies the SCL label where the method is implemented</td>
</tr>
<tr>
<td>signature</td>
<td>List</td>
<td>Input</td>
<td>specifies the id of a list containing named items that define the method signature</td>
</tr>
</tbody>
</table>

Details

The _addMethod signature that passes metadataList is valid on both class and instance; the remaining signatures are valid on instance only.
Object Class: _addReceiver

Adds an object to another object's _receivers_ list

---

Syntax

```
objectName_addReceiver( receiverID );
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>receiverID</td>
<td>Numeric</td>
<td>Input</td>
<td>specifies the identifier of the object to add to the list of broadcast recipients</td>
</tr>
</tbody>
</table>

Details

If the receivers list does not exist, the _addReceiver_ method creates it and then adds the specified object to the list. The _addReceiver_ method also adds the sending object to the receiving object's receiverID.

Use _addReceiver_ to add receiverID to the list of recipients that receive messages when object executes the _broadcast_ method.

---

Object Class: _attributesDialog

Displays the object's attributes in a mini-property window or in its custom property window if one has been assigned

---

Syntax

```
objectName_attributesDialog( );
objectName_attributesDialog( rc );
```

---

Object Class: _binit

Initializes an object at design time

---

Syntax

```
objectName_binit( );
```
Details
This method runs automatically when the object is created during design time. You can override \_binit to add processing that occurs during the creation of an object at design time.

Object Class: \_broadcast

Broadcasts a message to a list of objects

Syntax

objectName\_broadcast( message, numArg, charArg );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>Character</td>
<td>Input</td>
<td>specifies the text of the broadcast message that is passed to each receiver’s _receive method</td>
</tr>
<tr>
<td>numArg</td>
<td>Numeric</td>
<td>Input</td>
<td>specifies a number of an identifier of a list that contains broadcast data that is passed to each receiver</td>
</tr>
<tr>
<td>charArg</td>
<td>Character</td>
<td>Input</td>
<td>specifies a string that is passed to each receiver</td>
</tr>
</tbody>
</table>

Object Class: \_bterm

Terminates an object at design time

Syntax

objectName\_bterm( );

Details
This method runs automatically when the object is terminated at design time.
**Object Class: _changeAttribute**

Changes an attribute's metadata on a class or object

---

**Syntax**

```plaintext
return = objectName_changeAttribute( attributeName, metadataList );
```

**Argument Type Use Description**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>Numeric</td>
<td>Return</td>
<td>returns a value indicating whether the attribute was successfully changed: 0 if successful, 1 if not successful</td>
</tr>
<tr>
<td>attributeName</td>
<td>Character</td>
<td>Input</td>
<td>specifies the name of the attribute to change</td>
</tr>
<tr>
<td>metadataList</td>
<td>List</td>
<td>Input</td>
<td>specifies a list containing the named items for the attribute metadata</td>
</tr>
</tbody>
</table>

**Details**

When you use _changeAttribute to change a NEW attribute, any metadata item can be changed. However, when you use _changeAttribute to override an INHERITED attribute, the following rules apply:

- You cannot change NAME, TYPE, GETCAM, SETCAM, AUTOCREATE, or STATE. (SAS/AF will change STATE for you). If these items are passed via the metadata list, they are ignored.
- You can only change certain items to a more restrictive state, including LINKABLE (yes to no), EDITABLE (yes to no), and SCOPE (public to private or protected). If invalid values are passed for these items, they are ignored.
- You can change all other items.

---

**Object Class: _changeEvent**

Changes an event's metadata on a class or object

---

**Syntax**

```plaintext
return = objectName_changeEvent( eventName, metadataList );
```

```plaintext
return = objectName_changeEvent( eventName, metadataList, method );
```
Object Class: _changeEventHandler

Changes an event handler's metadata on a class or object

**Syntax**

```javascript
return = objectName_changeEventHandler(sender, event, action, metadataList);
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>Numeric</td>
<td>Return</td>
<td>returns a value indicating whether the event handler was successfully changed: 0 if successful, 1 if not successful</td>
</tr>
<tr>
<td>eventName</td>
<td>Character</td>
<td>Input</td>
<td>specifies the name of the event to change</td>
</tr>
<tr>
<td>metadataList</td>
<td>List</td>
<td>Input</td>
<td>specifies a list containing named items of the event metadata to change</td>
</tr>
<tr>
<td>method</td>
<td>Character</td>
<td>Input</td>
<td>specifies the method that triggers the event; used to uniquely identify the event to change in case there are multiple events by the same name</td>
</tr>
</tbody>
</table>
Object Class: _changeMethod

Changes a method's metadata on a class or object

Syntax

return = objectName_changeMethod( methodName, metadataList, returnMethodID );
return = objectName_changeMethod( methodID, metadataList, returnMethodID );

Details

If a method has only one signature, you can use either _changeMethod signature, _changeMethod(methodName) or _changeMethod(methodID). For methods that have multiple signatures, the following rules apply:

- You can change NAME, DESCRIPTION, SCOPE, and ENABLED items at the method name level using _changeMethod(methodName), which would affect all occurrences of that method.
You must use _changeMethod(methodID) with the appropriate id if you want to change SIGNATURE, ENTRY, or LABEL since these items correspond to a specific signature. If you try to use _changeMethod(methodName) to change one of these signature-dependent items, the action fails and you are prompted to use the correct signature.

- SCOPE can be different between signatures of the same method. Use _changeMethod(methodName) to change SCOPE for all signatures of a method, or use _changeMethod(methodID) to change SCOPE for a specific signature.

When changing a NEW method, any item can be changed if performed on a class ID instead of an instance. If you change an instance (per-instance) method, you cannot change the SCOPE; it is always PUBLIC on instance methods.

When using _changeMethod to override an INHERITED method:
- You cannot change NAME and SIGNATURE. If you pass these items, they are ignored.
- On a class ID, you can only change SCOPE to be less restrictive. (That is, if the inherited method is PROTECTED, you can only change it to PRIVATE. If the inherited method is PUBLIC, you can change it to either PRIVATE or PROTECTED.) You cannot change a PROTECTED inherited method to PUBLIC.

- On an instance, you cannot change SCOPE; it is always PUBLIC for inherited methods.
- Changing the value for ENABLED does not mark an INHERITED method as OVERRIDDEN. Changes to any other item, however, also changes the STATE from INHERITED to OVERRIDEN.
- If you forward the method, ENTRY= is ignored. Instead, COMPONENT and COMP_METHOD are the checked items.

SAS/AF software returns the new methodID of the method if you use _changeMethod to override an inherited method. For example, if someMethod is an inherited method whose id is 806 and _changeMethod is run to override it, the someMethod method is overridden and a new id is returned via the methodID argument.

---

**Object Class: _deleteAttribute**

Deletes an attribute from a class or object

---

**Syntax**

```
return = objectName_deleteAttribute( attributeName );
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributeName</td>
<td>Character</td>
<td>Input</td>
<td>specifies the name of the attribute to delete</td>
</tr>
</tbody>
</table>
Object Class: _deleteEvent

Deletes an event registered on a class or object

Syntax

\[
\text{return } = \text{objectName}_\text{deleteEvent}( \text{eventName} );
\]

\[
\text{return } = \text{objectName}_\text{deleteEvent}( \text{eventName}, \text{method} );
\]

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>Numeric</td>
<td>Return</td>
<td>returns a value indicating whether the event was successfully deleted: 0 if successful, 1 if not successful</td>
</tr>
<tr>
<td>eventName</td>
<td>Character</td>
<td>Input</td>
<td>specifies the name of the event to delete</td>
</tr>
<tr>
<td>method</td>
<td>Character</td>
<td>Input</td>
<td>specifies the name of the method that triggers the event. Used to uniquely identify which event to delete when multiple events by the same name exist</td>
</tr>
</tbody>
</table>

Object Class: _deleteEventHandler

Deletes an event handler registered on a class or object

Syntax

\[
\text{return } = \text{objectName}_\text{deleteEventHandler}();
\]

Object Class: _deleteInstanceMethod

Removes a per-instance method from a class or object
**Syntax**

```
objectName_deleteInstanceMethod( method );
objectName_deleteInstanceMethod( method, entryName );
objectName_deleteInstanceMethod( method, entryName, label );
objectName_deleteInstanceMethod( method, entryName, label, whenToExecute );
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>method</td>
<td>Character</td>
<td>Input</td>
<td>specifies the name of the per-instance method to delete</td>
</tr>
<tr>
<td>entryName</td>
<td>Character</td>
<td>Input</td>
<td>specifies the SAS/AF entry where the method code is implemented</td>
</tr>
<tr>
<td>label</td>
<td>Character</td>
<td>Input</td>
<td>specifies the SCL label where the method is implemented</td>
</tr>
<tr>
<td>whenToExecute</td>
<td>Character</td>
<td>Input</td>
<td>specifies when to execute this per-instance method in relation to others; user ‘BEFORE’, ‘OVERRIDE’ (default), or ‘AFTER’</td>
</tr>
</tbody>
</table>

---

**Object Class: _deleteLink**

Deletes an attribute link on an object

**Syntax**

```
return = objectName_deleteLink( attributeName );
return = objectName_deleteLink( attributeName, sourceObject, sourceAttribute );
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>Numeric</td>
<td>Return</td>
<td>returns a value indicating whether the link was successfully deleted: 0 if successful, 1 if not successful</td>
</tr>
<tr>
<td>attributeName</td>
<td>Character</td>
<td>Input</td>
<td>specifies the name of the attribute whose link you want to delete. If multiple links exist on the attribute, the last attribute defined is the one deleted.</td>
</tr>
</tbody>
</table>
Object Class: _deleteMethod

 Deletes a method on a class or object

Syntax

\[
\text{return} = \text{objectName} \_\text{deleteMethod}( \text{methodID} );
\]

\[
\text{return} = \text{objectName} \_\text{deleteMethod}( \text{methodName} );
\]

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>Numeric</td>
<td>Return</td>
<td>returns a value indicating whether the method was successfully deleted: 0 if successful, 1 if not successful</td>
</tr>
<tr>
<td>methodID</td>
<td>Numeric</td>
<td>Input</td>
<td>specifies the id of the method to delete. Useful when a method is overloaded to uniquely identify which method signature to delete</td>
</tr>
<tr>
<td>methodName</td>
<td>Character</td>
<td>Input</td>
<td>specifies the name of the method to delete</td>
</tr>
</tbody>
</table>

Details

SAS/AF software returns the new methodID of the method if you are trying to return an overridden method to its inherited state. For example, if someMethod is an overridden method whose id is 1207 and _deleteMethod is run to remove the override, the someMethod method is returned to its inherited state and a new id is returned via the methodID argument.
**Object Class: _deleteReceiver**

Deletes an object from another object’s list of recipients

---

### Syntax

```objectName_deleteReceiver( receiverID );```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>receiverID</td>
<td>Numeric</td>
<td>Update</td>
<td>specifies the identifier of the object to remove from the broadcast recipient’s list. It also removes the sending object from the receiving object’s <em>senders</em> list.</td>
</tr>
</tbody>
</table>

---

### Details

The _deleteReceiver method removes a receiverID from the broadcast recipients list of an object. Removing a receiver object from a sending object’s receivers list is useful when an object no longer needs to receive broadcasts from a particular sender. The _deleteReceiver method also removes the sending object, from the receiving object’s senders list.

---

**Object Class: _getAttribute**

Returns metadata for a specific attribute on a class or object

---

### Syntax

```return = objectName_getAttribute( attributeName, metadataList );```
```return = objectName_getAttribute( attributeName, metadataList, instances );```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>Numeric</td>
<td>Return</td>
<td>returns a value indicating whether the attribute was successfully retrieved: 0 if successful, 1 if not successful</td>
</tr>
<tr>
<td>attributeName</td>
<td>Character</td>
<td>Input</td>
<td>specifies the name of the attribute to retrieve</td>
</tr>
</tbody>
</table>
## Object Class: _getAttributes

Returns metadata for all attributes on a class or object

### Syntax

```
return = objectName_getAttributes( metadata );
return = objectName_getAttributes( metadata2, inherited );
return = objectName_getAttributes( metadata3, inherit, value );
```

### Argument Table

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>return</td>
<td>Numeric</td>
<td>Return</td>
<td>specifies the id of a list that contains sublists of attribute metadata</td>
</tr>
<tr>
<td>metadata</td>
<td>List</td>
<td>Output</td>
<td></td>
</tr>
<tr>
<td>metadata2</td>
<td>List</td>
<td>Update</td>
<td>specifies the list of sublists containing attribute metadata</td>
</tr>
<tr>
<td>inherited</td>
<td>Character</td>
<td>Input</td>
<td>specifies whether inherited attributes are returned in the metadata list: 'Yes' returns all metadata, 'No' does not return inherited attributes</td>
</tr>
<tr>
<td>metadata3</td>
<td>List</td>
<td>Update</td>
<td></td>
</tr>
</tbody>
</table>
### Argument Type Use Description

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inherit</td>
<td>Character</td>
<td>Input</td>
<td>specifies whether inherited attributes are returned in the metadata list</td>
</tr>
<tr>
<td>value</td>
<td>Character</td>
<td>Input</td>
<td>specifies whether the metadata list contains VALUE = as a named item. Only valid on the instance.</td>
</tr>
</tbody>
</table>

### Details

#### Object Class: `_getAttributeValue`

Returns the value of a specific attribute on an object

**Syntax**

```plaintext
return = objectName_getAttributeValue();
```

**Details**

When `_getAttributeValue` executes, it performs the following sequence of actions:

1. Determines if requested attribute is defined; returns 2 if attribute is undefined and processing ends.
2. Determines if value matches the attribute's type; returns 3 if it does not match and processing ends.
3. Executes getCAM if one is defined for the attribute; returns 7 if the CAM cannot execute, otherwise returns the value returned from the CAM.
4. Returns the value of the attribute.

If you add a getCAM to an attribute, `_getAttributeValue` returns the return value from your CAM as the value for `rc`.

#### Object Class: `_getClass`

Returns the identifier of an object's class

**Syntax**

```plaintext
objectName_getClass( classID );
```
Object Class: `_getData`

Returns the numeric or character value of an object

Syntax
return = objectName._getData();

Details
The `_getData` method is a convenient method for getting the numeric or character value of an object. You must pass in assignable values (variables or array element references) for `numVar` or `charVar`.

The default behavior is to call the object’s `_getText` method if the requested value is character; it calls `_getValue` if the requested value is numeric. Some objects override these methods to provide custom behavior.

Object Class: `_getEvent`

Returns metadata for a specific registered event on a class or object

Syntax
return = objectName._getEvent( eventName, metadata );
return = objectName._getEvent( eventName, metadata, method );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eventName</td>
<td>Character</td>
<td>Input</td>
<td>specifies the name of the event to retrieve</td>
</tr>
<tr>
<td>Return</td>
<td>Numeric</td>
<td>Return</td>
<td>returns a value indicating whether the event was successfully retrieved: 0 if successful, 1 if not successful</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>classID</td>
<td>Numeric</td>
<td>Output</td>
<td>returns the identifier of the object’s class</td>
</tr>
</tbody>
</table>
### Argument Type Use Description

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metadata</td>
<td>List</td>
<td>Update</td>
<td>specifies the list returned containing the named items for the event metadata</td>
</tr>
<tr>
<td>method</td>
<td>Character</td>
<td>Input</td>
<td>specifies the method name associated with eventName. This is required when multiple events exist by the same name to uniquely identify the event to retrieve</td>
</tr>
</tbody>
</table>

### Details

**Object Class: _getEventHandler**

Returns metadata for a specific event handler on a class or object

#### Syntax

```
return = objectName_getEventHandler( sender, event, action, metadataList );
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>Numeric</td>
<td>Return</td>
<td>returns a value indicating whether the event handler was successfully retrieved: 0 if successful, 1 if not successful</td>
</tr>
<tr>
<td>sender</td>
<td>Character</td>
<td>Input</td>
<td>specifies the object that sends the event</td>
</tr>
<tr>
<td>event</td>
<td>Character</td>
<td>Input</td>
<td>specifies the event that triggers the action</td>
</tr>
<tr>
<td>action</td>
<td>Character</td>
<td>Input</td>
<td>specifies the action to take when the event occurs</td>
</tr>
<tr>
<td>metadataList</td>
<td>List</td>
<td>Update</td>
<td>returns the list containing named items that represent the event handler metadata</td>
</tr>
</tbody>
</table>

### Details

**Object Class: _getEventHandlers**

Returns metadata for all event handlers on a class or object
**Object Class: _getEvents**

Returns metadata for all registered events on a class or object

---

**Syntax**

```c
return = objectName_getEventHandler( metadata );
return = objectName_getEventHandler( metadata, inherited );
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>Numeric</td>
<td>Return</td>
<td>returns a value indicating whether the event handlers were successfully retrieved: 0 if successful, 1 if not successful</td>
</tr>
<tr>
<td>metadata</td>
<td>List</td>
<td>Update</td>
<td>returns the list containing the event handler metadata</td>
</tr>
<tr>
<td>inherited</td>
<td>Character</td>
<td>Input</td>
<td>specifies whether inherited event handlers are returned in the metadata list: 'Yes' includes INHERITED event handlers, 'No' does not</td>
</tr>
</tbody>
</table>

---

**Object Class: _getEvents**

Returns metadata for all registered events on a class or object

---

**Syntax**

```c
return = objectName_getEvents( metadata );
return = objectName_getEvents( metadata, inherited );
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>Numeric</td>
<td>Return</td>
<td>returns a value indicating whether the events where successfully retrieved: 0 if successful, 1 if not successful</td>
</tr>
<tr>
<td>metadata</td>
<td>List</td>
<td>Update</td>
<td>returns the list with sublists containing event metadata</td>
</tr>
<tr>
<td>inherited</td>
<td>Character</td>
<td>Input</td>
<td>specifies whether inherited events are returned in the metadata list: 'Yes' returns INHERITED events, 'No' does not</td>
</tr>
</tbody>
</table>
Object Class: _getMethodList

Returns the listid of per-instance methods for the object

---

Syntax

```
objectName._getMethodList( instanceMethods );
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>instanceMethods</td>
<td>Numeric</td>
<td>Update</td>
<td>specifies the identifier of an SCL list to contain the per-instance methods</td>
</tr>
</tbody>
</table>

Details

The _getMethodList returns the list id of per-instance methods for object. The list is in the same format as returned by the _getMethodList on class objects, except for per-instance methods that have had BEFORE or AFTER methods defined. In such cases, the method specification is a sublist of BEFORE, OVERRIDE, and AFTER per-instance methods.

Object Class: _getLink

Returns the attribute links associated with a specific attribute on an object

---

Syntax

```
return = objectName._getLink( attributeName, links );
```
**Object Class: _getMethod**

Returns the metadata for a specific method on a class or object

---

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>Numeric</td>
<td>Return</td>
<td>returns a value indicating whether the attribute links were successfully retrieved: 0 if successful, 1 if not successful</td>
</tr>
<tr>
<td>attributeName</td>
<td>Character</td>
<td>Input</td>
<td>specifies the name of the attribute for which you want to retrieve links</td>
</tr>
<tr>
<td>links</td>
<td>List</td>
<td>Update</td>
<td>specifies the identifier of a list that when returned is filled with attribute link information</td>
</tr>
</tbody>
</table>

**Details**
The _getLink method is only available for instances; you can only link on the instance.

---

**Object Class: _getLinks**

Returns a list of all attribute links on an object

**Syntax**

```
return = objectName_getLinks( links );
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>Numeric</td>
<td>Return</td>
<td>returns a value indicating whether the attribute links were successfully retrieved: 0 if successful, 1 if not successful</td>
</tr>
<tr>
<td>links</td>
<td>List</td>
<td>Update</td>
<td>specifies the identifier of a list that when returned contains all attribute links on the object</td>
</tr>
</tbody>
</table>

**Details**
The _getLinks method retrieves a list of sublists where named items on the list represent the names of the attributes that have defined links. Each sublist, then, contains the links defined for that attribute.

---

**Object Class: _getMethod**

Returns the metadata for a specific method on a class or object
### Syntax

```plaintext
return = objectName_getMethod( methodName, metadata );
return = objectName_getMethod( methodID, metadata2 );
return = objectName_getMethod( methodName, metadata3, instance );
return = objectName_getMethod( methodID, metadata4, instances );
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>Numeric</td>
<td>Return</td>
<td>returns a value indicating whether the method was successfully retrieved: 0 if successful, 1 if not successful</td>
</tr>
<tr>
<td>methodName</td>
<td>Character</td>
<td>Input</td>
<td>specifies the name of the method you want to retrieve</td>
</tr>
<tr>
<td>metadata</td>
<td>Numeric</td>
<td>Update</td>
<td>specifies the identifier of a list that when returned contains method metadata. If the method is overloaded, the list will contain a sublist for each signature.</td>
</tr>
<tr>
<td>methodID</td>
<td>Numeric</td>
<td>Input</td>
<td>specifies the unique id of the method you want to retrieve. Useful when a method is overloaded to uniquely identify the method.</td>
</tr>
<tr>
<td>metadata2</td>
<td>List</td>
<td>Update</td>
<td>specifies the identifier of a list that when returned contains method metadata</td>
</tr>
<tr>
<td>metadata3</td>
<td>List</td>
<td>Update</td>
<td>specifies the identifier of a list that when returned is filled with the method metadata</td>
</tr>
<tr>
<td>instance</td>
<td>Character</td>
<td>Input</td>
<td>specifies whether methods are returned for instances of the class. This option is only valid for class instances.</td>
</tr>
<tr>
<td>metadata4</td>
<td>List</td>
<td>Update</td>
<td>specifies the identifier of a list that when returned is filled with method metadata</td>
</tr>
<tr>
<td>instances</td>
<td>Character</td>
<td>Input</td>
<td>specifies whether the method is returned for instances of the class. This option is valid for class instances only.</td>
</tr>
</tbody>
</table>

### Details

Consider the following information when you use _getMethod:
Use the _getMethod signature that passes the instance argument only with class IDs; use the other _getMethod signatures with either class ID or instance.

To retrieve the metadata for a specific signature, use the form _getMethod(methodID). If you use the method name as the argument, you retrieve the first signature in the list.

_getMethod does not return a method in an ancestor class that is out of scope (that is, defined as private or protected) for the calling object or class. You can use _getIMethodList if you need to return all methods regardless of scope.

---

**Object Class: _getMethods**

Returns the metadata for all methods on a class or object

**Syntax**

\[
\text{return } = \text{objectName}_\text{getMethods} ( \text{metadata} ); \\
\text{return } = \text{objectName}_\text{getMethods} ( \text{metadata, inherited} );
\]

**Argument** | **Type** | **Use** | **Description**
--- | --- | --- | ---
Return | Numeric | Return | returns the methodID that uniquely identifies the specified method
methodName | Character | Input | specifies the name of the method
sigString | Character | Input | specifies the signature string for the method
signatureList | List | Input | specifies the signature list for the method

---

**Object Class: _getMethods**

Returns the method ID for a given method name and signature

**Syntax**

\[
\text{return } = \text{objectName}_\text{getMethodsID} ( \text{methodName, sigString} ); \\
\text{return } = \text{objectName}_\text{getMethodsID} ( \text{methodName, signatureList} );
\]

**Argument** | **Type** | **Use** | **Description**
--- | --- | --- | ---
Return | Numeric | Return | returns the methodID that uniquely identifies the specified method
methodName | Character | Input | specifies the name of the method
sigString | Character | Input | specifies the signature string for the method
signatureList | List | Input | specifies the signature list for the method
Object Class: _getModelID

Returns the identifier of the associated model for an object

Syntax

```
return = objectName._getModelID();
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>Numeric</td>
<td>Return</td>
<td>returns a value indicating whether the methods were successfully retrieved: 0 if successful, 1 if not successful</td>
</tr>
<tr>
<td>metadata</td>
<td>List</td>
<td>Update</td>
<td>specifies the identifier of a list that when returned is filled with sublists containing method metadata for each method</td>
</tr>
<tr>
<td>inherited</td>
<td>Character</td>
<td>Input</td>
<td>specifies whether inherited methods are returned in the metadata list: 'Yes' indicates INHERITED methods will be included.</td>
</tr>
</tbody>
</table>

Details

If you run _getMethods against an instance of a class, the method only returns the PUBLIC methods associated with the object. If you run _getMethods on a class ID, the method returns all methods except any PRIVATE methods the parent may have.

For methods that support multiple signatures, _getMethods returns a separate sublist for each signature. For example, if a method called someMethod has two signatures, _getMethods returns two sublists named someMethod, but each has a different signature and unique method ID.

_getMethods does not return methods from an ancestor class that are out of scope (that is, defined as private or protected) for the calling object or class. You can use _getIMethodList if you need to return all methods regardless of scope.
**Object Class: _getProperties**

Fills a list with properties of an object

---

**Syntax**

objectName._getProperties( properties );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td>Numeric</td>
<td>Update</td>
<td>specifies the identifier of a list that when returned is filled with object properties</td>
</tr>
</tbody>
</table>

**Details**

The _getProperties method combines the results of _getAttributes, _getEvents, _getEventHandlers, and _getMethods into a single method.

---

**Object Class: _getSignatureString**

Generates a method signature string given a specific method signature list

---

**Syntax**

return = objectName._getSignatureString( methodID );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>Character</td>
<td>Return</td>
<td>returns the signature string for the specified method</td>
</tr>
<tr>
<td>methodID</td>
<td>Numeric</td>
<td>Input</td>
<td>specifies the methodID that uniquely identifies the method</td>
</tr>
</tbody>
</table>

---

**Object Class: _hasMethod**

Determines whether a method exists for an object
Syntax

objectName_hasMethod( methodName, status );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>methodName</td>
<td>Character</td>
<td>Input</td>
<td>specifies the method</td>
</tr>
<tr>
<td>status</td>
<td>Numeric</td>
<td>Output</td>
<td>returns a value indicating whether the method exists for the object: 1 if it exists, 0 if it does not exist</td>
</tr>
</tbody>
</table>

Details

A method exists if it is defined in the object’s instance methods (see _setInstanceMethod), in the object’s class, or in an object name in a delegate’s list for the object’s class or one of the ancestor classes. The method may be inherited.

Object Class: _help

Invokes help for the object

Syntax

objectName_help( );

Object Class: _init

Initializes an object

Syntax

objectName_init( );
objectName_init( numArg );

Details

When an object is created, SAS/AF software automatically invokes the _init method; you do not call _init explicitly unless you write your own _init method to override...
default processing. In this case, you must invoke the parent class _init method so the object is initialized according to the parent class' specifications. When calling the inherited _init method, pass _SELF_ as object. _SELF_ is a special variable containing the ID of the current object.

If you override the _init method, you must call the super _init method (the _init method inherited from the parent class) before any other methods can be invoked on the object. The object is not fully instantiated until the _init method for the object's parent class is executed.

A numeric argument can be sent to the _init method by either the _new method or the SCL function INSTANCE.

---

**Object Class: _initconstructor**

**Syntax**

```plaintext
objectName_initconstructor();
```

---

**Object Class: _isA**

Determines whether an object is a member of a class

**Syntax**

```plaintext
objectName_isA( className, descendent );
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>className</td>
<td>Character</td>
<td>Input</td>
<td>specifies the one-, two- or three-level name of a class</td>
</tr>
<tr>
<td>descendent</td>
<td>Numeric</td>
<td>Output</td>
<td>returns a value indicating whether the object is a descendent of the specified class: 1 is a descendent, 0 is not a descendent</td>
</tr>
</tbody>
</table>

**Details**

The _isA method determines whether an object is a member of a particular class (or a subclass of a class) by comparing the one-, two-, or three-level class name to the value specified in className (ignoring case). This method returns a value of 1 (true) if the class name or the name of any of its ancestor classes ends in the specified value.
This check is useful if you wish to write generic code that can be invoked on any object but that processes the code in a specific manner if the object is a member of a particular class.

You can specify className as a two-level name to compare the library and catalog for the class and determine whether a class exists in a particular catalog.

---

**Object Class: _new**

Invoked automatically when an object is created using the _new method from the Class class or the _new_ operator in SCL

**Syntax**

```plaintext
return = objectName._new();
```

**Details**

When the _new method is invoked on an object, a new instance is created. The _new method of the new instance is automatically invoked. By default, the _new method of the Object class has no behavior. However, you can override the _new method in subclasses you create, and use your _new method to initialize your object with parameters that you pass.

---

**Object Class: _newClass**

Creates a new class based on the current instance

**Syntax**

```plaintext
return = objectName._newClass(entryName);
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>Object</td>
<td>Return</td>
<td>returns the identifier of the new class created from the instance</td>
</tr>
<tr>
<td>entryName</td>
<td>Character</td>
<td>Input</td>
<td>specifies the four-level name of the CLASS entry to create</td>
</tr>
</tbody>
</table>
Object Class: _onAttributeChange

Invoked automatically when an attribute changed event occurs to handle attribute linking

Syntax
return = objectName_onAttributeChange( );

Details
The _onAttributeChanged method is called internally as the event handler that handles the attribute linking that happens when all 'attribute-name Changed' events occur. It receives the object, reads the information it needs, then calls the _setAttributeValue method.

Object Class: _onContentsUpdated

Virtual method for 'contents changed' event handler

Syntax
return = objectName_onContentsUpdated( );

Details
This virtual method is used in model/view component communication. Using the contentsUpdatedAttribute attribute, a model can list one or more attributes that are considered “critical,” which means that if any of those attribute values change, the attached viewer should be notified. When an attribute value is changed, the _setAttributeValue method

1 determines whether the attribute is stored in the contentUpdatedAttributes list
2 determines whether there is a model attached to the view
3 and, if both conditions are true, sends the 'contents updated' event.

The viewer’s event handler listens for the 'contents updated' event and runs its _onContentsUpdated method.

When you design a viewer to work with a model, you can override this method and add the functionality that updates the viewer.

Object Class: _printObject

Used for subclassing
**Object Class: _purgeEventHandlers**

Removes all event handlers that identify this object as either the sender or recipient of an event.

**Syntax**

```
return = objectName_purgeEventHandlers( );
```

---

**Object Class: _receive**

Receives a broadcast message from another object.

**Syntax**

```
objectName_receive( message, numArg, charArg, senderID );
```
### Argument Type Use Description

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>Character</td>
<td>Input</td>
<td>specifies the message broadcast from the sender through the _broadcast method</td>
</tr>
<tr>
<td>numArg</td>
<td>Numeric</td>
<td>Input</td>
<td>specifies the numeric argument broadcast from the sender</td>
</tr>
<tr>
<td>charArg</td>
<td>Character</td>
<td>Input</td>
<td>specifies the character argument broadcast from the sender</td>
</tr>
<tr>
<td>senderID</td>
<td>Numeric</td>
<td>Input</td>
<td>specifies the object identifier of the sender</td>
</tr>
</tbody>
</table>

### Details

When a message is sent from one object to another with the _broadcast method, SAS/AF software invokes the _receive method of each object in the sender's _receivers list. This is true only when the value of the charMsg parameter being broadcast is the name of an item on the recipient's _receive list or if the receiver or the receiver's class does not have a _receive list.

The numArg and charArg passed from the sender via the _broadcast method are forwarded to the receiver via the same parameters in the _receive parameters list, and the object identifier of the sender is passed as well, allowing the receiver to query the sender for additional data if necessary.

You can use an SCL list to pass a number of values from the sender to the recipient by specifying the list identifier in numArg.

**Note:** It is recommended that you implement component communication using attribute linking, event handling, or model/view.

---

**Object Class: _resetEventHandlers**

Internal method

**Syntax**

```
return = objectName_resetEventHandlers( );
```

---

**Object Class: _sendEvent**

Declare that an event has occurred and take all actions specified to occur by _setEventHandler or _addEventHandler
Object Class: _setAttributeValue

Sets the value for a specific attribute on an object

Syntax

return = objectName_sendEvent( );

Details

When _setAttributeValue executes, it performs the following sequence of actions:

1. Determines if requested attribute is defined; returns 2 if attribute is undefined and processing ends.
2. Determines if value matches the attribute's type; returns 3 if it does not match and processing ends.
3. Determines value of attribute metadata item EDITABLE; returns 5 if 'No' and processing ends.
4. If attribute has a specified VALIDVALUES item, determines whether value is found in list (or specified SLIST entry) or executes the SCL entry defined for VALIDVALUES; returns 4 if value is not in VALIDVALUES.
   Processing continues to setCAM.
5. If attribute has a specified VALIDVALUES item, returns 7 if the CAM cannot execute, otherwise returns the value returned from the CAM.
6. Sets the value for the attribute.
7. Refreshes the region if object is a visual control.
8. If attribute metadata item SENDEVENT='Yes', _setAttributeValue:
   - executes the _bAttributeChanged method at design time (if running in a frame context)
   - sends the 'attribute-name Changed' event
   - checks the contentsUpdatedAttributes attribute and sends the 'contents updated' event if the attribute is listed

If you add a setCAM to an attribute, _setAttributeValue returns the return value from your CAM as the value for rc.
Object Class: _setData

Sets the numeric or character value of an object

Syntax
return = objectName_setData( );

Details
The _setData method is a convenient method for setting the numeric or character value of an object.

The default behavior of these signatures is to call the object’s _setText method if the requested value is character (that is, a character value is supplied for charValue); it calls _setValue if the requested value is numeric (that is, a numeric value is supplied for numValue). Some objects override these method to provide custom behavior.

Object Class: _setEventHandler

Specify the action that takes place when a particular event occurs

Syntax
return = objectName_setEventHandler( );
Details

Table 4.1  Values for Sender

<table>
<thead>
<tr>
<th>Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>an object ID</td>
<td>N</td>
<td>sender is an object ID of an object</td>
</tr>
<tr>
<td>*</td>
<td>C</td>
<td>sender is any sender. This declares that the action should take place whenever the named event occurs, regardless of the sender of the event.</td>
</tr>
<tr>
<td>NAME</td>
<td>C</td>
<td>sender is a valid widget name; the name of a widget on the same frame as the widget identified in object-id. NAME can only be used if the action is sent to a widget on a frame.</td>
</tr>
</tbody>
</table>

Object Class: _setInstanceMethod

Adds or changes a per-instance method of a class or object

Syntax

objectName_setInstanceMethod( methodName, entry );
objectName_setInstanceMethod( methodName, entry, label );
objectName_setInstanceMethod( methodName, entry, label, when );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>methodName</td>
<td>Character</td>
<td>Input</td>
<td>specifies the method name</td>
</tr>
<tr>
<td>entry</td>
<td>Character</td>
<td>Input</td>
<td>specifies the SAS/AF entry where the method code is implemented</td>
</tr>
</tbody>
</table>
Object Class: _showPropertiesWindow

Displays the properties window for the object

Syntax

```
objectName_showPropertiesWindow( );
objectName_showPropertiesWindow( rc );
```
Object Class: _term

Deletes an object

Syntax

objectName._term();

Details

To perform additional processing when you delete an object (for example, when you
close any open data set or delete nested lists) you can write your own _term method to
override the default _term method. However, if you do this, you should also invoke the
super _term method (the _term method inherited from the parent class) so the object is
deleted according to the parent class’ specifications. Usually this should be the last step
in your SCL method. After the super _term method is invoked, the object does not exist
anymore. The object identifier is then invalid and the object cannot receive broadcast
messages or methods.

When a FRAME application ends, the widgets and FRAME objects are deleted by
running their _term methods, even if you override the _term method and omit a call to
the _term method using the SUPER routine.

You do not have to override the _term method to delete most sublists of an object. If
you define an instance variable in a class such as type LIST, sublist instance variables
are tagged during object creation with a new list attribute, AUTODELETE. When the
Object class _term method executes, it deletes the object with a new AUTODELETE
option for DELLIST. When this option is used for DELLIST, any sublists that have the
AUTODELETE option set are also deleted. This differs from the RECURSIVE option of
DELLIST, since only sublists with AUTODELETE are affected. AUTODELETE is
ignored if the sublist has the NODELETE attribute.

The default behavior for DELLIST is NOAUTODELETE, so existing use of the
DELLIST function is not affected.

You may also set the AUTODELETE list attribute on other lists that you add to your
object. For example, if your _init method allocates a list for the automatic instance
variable SUBLIST, you can designate that list to be automatically deleted, as shown
here:

if sublist then /* delete existing sublist */ rc =
dellist(sublist, ‘y autodelete’); sublist = makelist(); rc =
setlattr(sublist, ‘AUTODELETE’);

You can test for the AUTODELETE attribute with HASATTR:

hasAutoDelete = hasattr(listid, ‘AUTODELETE’);

You can turn it off with

rc = setlattr(listid,
‘NOAUTODELETE’);

COPYLIST and SAVELIST preserve the AUTODELETE attribute. This means that
if you execute COPYLIST for a list that is held in an object’s instance variable, the
COPY has the AUTODELETE attribute (as will any of its sublists, if the original
sublists had AUTODELETE). If you assign that COPY to another instance variable of
another object, for example, that list and its contents are deleted automatically when the second object is deleted.

For example, consider an object A with an instance variable LIST that contains the sublists S1 and S2. Object B is another object. Assume the objects' class define methods 'getSublist' and 'setSublist', which get and set the LIST instance variable.

```
a.getSublist(list); /* fetches inst var
LIST */
copy = copylist(list); /* sets inst var LIST*/
... b._term;
```

Since the list was not copied recursively, the list COPY (which has the AUTODELETE attribute) contains the sublists S1 and S2. When object B is deleted, its sublist LIST (which is the list COPY) is also deleted. Further, its sublists S1 and S2 are deleted, since they also have the AUTODELETE attribute. This situation is not common and is actually discouraged.

Another situation to consider but that should not be a problem is late deletion of automatic list instance variables. For example, if your class has an automatic list instance variable SUBLIST, you may have written your _term method as follows:

```
_term : method; call super(_self_, '_term_'); rc =
dellist(sublist); endmethod;
```

Now, the sublist is deleted automatically during the SUPER call so the DELLIST returns a nonzero value in RC since SUBLIST is not a valid listid. Normally, the SUPER _term is the last action of a _term method, so this should not be a major problem. Deleting the list before the SUPER _term is still fine but not always necessary.

The AUTOTERM feature exists to automatically invoke _term on all undeleted objects when an application terminates. The _term method will be sent to all objects in an unspecified order.

On the AF or AFAPPLICATION commands, use the AUTOTERM option to control the feature:

```
AF C=lib.cat.member.name
AUTOTERM=option AFA C=lib.cat.member.name AUTOTERM=option
```

You can also use the AFSYS command once an application is running:

```
AFSYS AUTOTERM option
```

**Option** is one of the following:

- **ON** enables the AUTOTERM features. AUTOTERM is on by default. Use this to enable it if it has been turned off.
- **OFF** disables the AUTOTERM feature. The software will not invoke _term on objects at task termination time.
- **VERBOSE** prints a NOTE: and dumps the object list of each object that still exists at task termination time. This works even if AUTOTERM is OFF; it serves as a debugging aid to identify which objects still exist but whose _term method has not run.
- **NOVERBOSE** disables the VERBOSE option. NOVERBOSE the default.

You cannot combine options in one string; use separate AUTOTERM= options on the command line or use separate AFSYS commands.
Object Class: _toString

Returns a string used to represent an object's value

Syntax

\[ \text{return = objectName}_\text{toString}(); \]

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>Character</td>
<td>Return</td>
<td>returns a character string describing the object</td>
</tr>
</tbody>
</table>

Object Class: _typeOf

Returns the type of an object

Syntax

\[ \text{objectName}_\text{typeof}( \text{type} ); \]

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Character</td>
<td>Output</td>
<td>returns the uppercase name of the object class subsystem</td>
</tr>
</tbody>
</table>

Object Class Events

Events specified for the Object Class class are described here.

- CBTFramename changed
  - Occurs when the CBTFramename attribute is changed
- attachedInterface changed
  - Occurs when the attachedInterface attribute is changed
- contents updated
  - Occurs when the value of an attribute in the contentsUpdatedAttributes is changed
contentsUpdatedAttributes changed
  Occurs when the contentsUpdatedAttributes attribute is changed
defaultAttribute changed
  Occurs when the defaultAttribute attribute is changed
help changed
  Occurs when the help attribute is changed
model changed
  Occurs when the model attribute is changed
name changed
  Occurs when the name attribute is changed