Objects

Version 2.0

GEOS Software Development Kit Library

Version 2.0

Objects

Initial Edition, Unrevised and Unexpanded

Geoworks, Inc.
Alameda, CA

Geoworks provides this publication “as is” without warranty of any kind, either express or implied, including but not limited to the implied warranties of merchantability or fitness for a particular purpose. Geoworks may revise this publication from time to time without notice. Geoworks does not promise support of this documentation. Some states or jurisdictions do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

Objects

Volume 1

Volume 2

Volume 3

List of C Examples

List of Figures

System Classes

1

GEOS provides three high-level classes which operate mostly behind the scenes. Your application will never include an object of any of these classes, though it will include objects of their subclasses. **MetaClass** is the root of the GEOS class tree and therefore handles many messages you may never realize get handled by your objects. **ProcessClass** is the class that handles creation of a primary thread for a process geode. **GenProcessClass**, a subclass of **ProcessClass**, handles additional application process behavior such as state saving and restoration as well as application opening and closing.

1.1 MetaClass

MetaClass is the ancestor class of every GEOS object. **MetaClass** is the location of basic messages and their handlers. Basic functionality for all objects—instantiation, initialization, detach, and destruction—is implemented within this class.

The only instance data field defined for **MetaClass** is the object's class pointer (the field is named *MI_base*). **MetaClass** has no other inherent data fields. You will never need to access the class pointer directly.

MetaClass also serves as a sort of placeholder for certain messages. Applications writers familiar with mouse input, for instance, know that MSG_META_START_SELECT signals a mouse click. This message is actually defined at the **MetaClass** level, though **MetaClass** itself has no handler for it. However, by defining this message at the Meta level, all classes which should be able to handle mouse events agree on the message number corresponding to a click. Normally, these messages are defined in the appropriate library, then exported. Applications and libraries using these messages can import them. For information about importing and exporting messages, see "GEOS Programming," Chapter 5 of the Concepts Book.

1.1.1 Special Messages

MSG_META_NULL, MSG_META_DUMMY

The following two messages are mainly place holders. They ensure that no message will have the value zero or one. (When a message is called, a value of zero equates to null.) These are not used by applications or objects in general.

1.1

■ MSG_META_NULL

```
void      MSG_META_NULL();
```

This message has no handler and is unused. It essentially ensures that no other message will ever have the value zero.

Interception: Don't.

■ MSG_META_DUMMY

```
void      MSG_META_DUMMY();
```

This message has no handler. You should not subclass this message to provide one. Certain object mechanisms, such as the resolution of a variant class, are activated by the object's receipt of MSG_META_DUMMY.

Interception: Don't.

1.1.2 Utility Messages

MetaClass also provides a number of other messages that are used throughout the system. These messages have been separated into loosely-defined categories and listed in the sections below.

1.1.2.1 Object Creation and Destruction

These messages handle creation, destruction, and initialization of all objects. The function and use of many of these messages are given in "GEOS Programming," Chapter 5 of the Concepts Book.

■ MSG_META_INITIALIZE

```
void        MSG_META_INITIALIZE();
```

Every object class should provide a handler for this message which should call the superclass and then perform any initialization of the instance data required.

Note that **GenClass** and **VisClass** have a default handler that sets up the Gen and Vis parts automatically.

Source: Object system itself, often in the middle of attempting to deliver another message to an object that hasn't yet been initialized.

Destination: Object whose instance data is not yet initialized.

Parameters: None.

Return: Nothing.

Interception: Any class wishing to have default instance data values other than all zeros should intercept this message to fill in the initial values for its instance data. For classes other than master classes, standard procedure is to call the superclass first, then perform any additional instance data initialization necessary. Master classes should *not* call the superclass, as MSG_META_INITIALIZE is unique among messages in that it is sent only to classes within the particular master group that needs to be initialized. Handlers of MSG_META_INITIALIZE should limit their activities to just stuffing instance data—specifically, object messaging is not allowed (though scanning vardata is OK). Object classes that inherit instance data (all but **MetaClass**) should call MSG_META_INITIALIZE on their superclass to initialize that portion of their instance data. In addition, they must initialize their own portion of the instance data (start by assuming it's all zeros). The order won't matter, so long as the handler doesn't depend on the inherited instance data having any particular value. When in doubt, call superclass first.

■ MSG_META_ATTACH

```
void        MSG_META_ATTACH();
```

This message is used for two different purposes: It can be sent to any geode that has a process aspect when the geode is first loaded. It can also be sent in the object world to notify objects on an "active list" that the application has been brought back up from a state file. As the method is used for different purposes, the data passed varies based on usage. Because of this difference in parameters, normally C applications will use one of the aliases for this message (MSG_META_ATTACH_PROCESS, MSG_META_ATTACH_THREAD, MSG_META_ATTACH_GENPROCESSCLASS, MSG_META_ATTACH_OBJECT, and MSG_META_ATTACH_GENAPPLICATION, each described below.)

■ MSG_META_ATTACH_PROCESS

1.1

```
@alias (MSG_META_ATTACH)
 void     MSG_META_ATTACH_PROCESS(
          word   value1,
          word   value2);
```

This message is sent to any geode which has a process when the geode is first loaded. By default, the handler for this message does nothing.

Source: **GeodeLoad()** kernel routine.

Destination: Newly created Process object (but not GenProcess object).

Parameters: *value1* Upper half of **GeodeLoad()** *appInfo* argument.

value2 Lower half of **GeodeLoad()** *appInfo* argument.

Return: Nothing.

Interception: No default handling provided, so if you are spawning an extra process and that process needs to do some initialization, then intercept this message.

■ MSG_META_ATTACH_GENPROCESSCLASS

```
@alias (MSG_META_ATTACH)
 void     MSG_META_ATTACH_GENPROCESSCLASS(
          MemHandle appLaunchBlock);
```

This message is sent to the GenProcess object when the geode is first loaded. By default, the handler for this message calls MSG_PROCESS_STARTUP_UI_THREAD, which checks to see if there are nay resources of the application which are marked as “ui-object” (they are marked this way in the **.gp** file), that is, to be run by a UI thread. If so, it then calls MSG_PROCESS_CREATE_UI_THREAD to create that thread, then marks the “ui-object” blocks as run by that thread.

The handler next calls one of the following messages:

MSG_GEN_PROCESS_OPEN_APPLICATION
For applications which are being started up regularly (that is, not restoring from a state file) and will appear on screen.

MSG_GEN_PROCESS_OPEN_ENGINE
For those applications that will operate in engine mode (i.e. non-visual).

MSG_GEN_PROCESS_RESTORE_FROM_STATE
For applications which are restoring from state. This is the case for applications that were running at the previous shutdown.

Source: **GeodeLoad()** kernel routine.

Destination: Newly created **GenProcessClass** (or subclass thereof) object.

Parameters: *appLaunchBlock* Block handle to block of structure **AppLaunchBlock**.

Return: Nothing.

Note that the passed **AppLaunchBlock** is destroyed.

Interception: No default handling provided, so if you are spawning an extra process and that process needs to do some initialization, then intercept this message.

■ MSG_META_ATTACH_GENAPPLICATION

```
@alias (MSG_META_ATTACH)
 void     MSG_META_ATTACH_PROCESS(
          MemHandle bh1,
          MemHandle bh2);
```

This message is sent to the GenApplication object by **GenProcessClass** when the application starts up (either for the first time, or when being restored from a state file).

Source: GenProcess object.

Destination: GenApplication object.

Parameters: *bh1* Block handle to block containing **AppLaunchBlock** parameters.

bh2 Extra state block from state file, or NULL if none. This is the same block as returned from

MSG_GEN_PROCESS_CLOSE_APPLICATION in some previous detach operation.

Return: Nothing.

Note that the **AppLaunchBlock** is destroyed.

Interception: Not generally required, since the default handler broadcasts the message out to everything on the application's active lists. This act causes the interface for the application to come up on screen.

1.1

■ MSG_META_ATTACH_OBJECT

```
@alias (MSG_META_ATTACH)
 void     MSG_META_ATTACH_OBJECT(
          word              flags,
          MemHandle         appLaunchBlock,
          MemHandle         extraState);
```

This message is sent to any object on the GenApplication object's active lists, or on one of those object's active lists. Note that this will not happen until the GenApplication is set usable by the GenProcess object.

Source: **GenApplicationClass** object.

Destination: Any object.

Parameters:

flags Flags providing state information.

appLaunchBlock Handle of **AppLaunchBlock**, or NULL if none.

extraState Handle of extra state block, or NULL if none. This is the same block as returned from MSG_GEN_PROCESS_CLOSE_APPLICATION, in some previous detach.

Return: Nothing.

Interception: Standard UI objects defined as needing to be placed on an active list will intercept this message to do whatever it is that they needed to do when the application is first loaded. Objects intercepting this message should call the superclass, in case it expects to receive this notification itself.

Warnings: If the specific UI uses this mechanism, then the **GenProcessClass** will have already destroyed the **AppLaunchBlock** and extra state block by the time the MSG_META_ATTACH is sent to objects on its active list.

◆Objects

■ MSG_META_ATTACH_THREAD

```
@alias (MSG_META_ATTACH)
 void    MSG_META_ATTACH_THREAD();
```

This message is sent to any thread spawned by MSG_PROCESS_CREATE_EVENT_THREAD.

Source: Kernel.

Destination: Newly created thread, specifically the class designated to handle the thread's messages (a subclass of **ProcessClass**).

1.1

Parameters: None.

Return: Nothing.

Interception: No default handling provided, so if you are spawning an extra process and that process needs to do some initialization, then intercept this message.

■ MSG_META_APP_STARTUP

```
void     MSG_META_APP_STARTUP(
         MemHandle          appLaunchBlock);
```

This message is related to MSG_META_ATTACH; the message is sent by the generic UI to the GenApplication object before it sends MSG_META_ATTACH to it. MSG_META_ATTACH is only sent when the application is becoming available to the user; if an application should be opened as a server without presenting any UI to the user, MSG_META_APP_STARTUP will be the only message sent to the application object upon start-up.

The default handler for this message will pass it on to all members of the MGCNLT_APP_STARTUP list.

Source: **GenProcessClass**; forwarded by **GenApplicationClass** to other objects. This message is sent upon application start-up before the UI for an application has been attached.

Destination: Any object that needs to be notified when the application is launched, regardless of whether the user will be interacting with the application.

Parameters: *appLaunchBlock* Handle of an **AppLaunchBlock**.

Return: The **AppLaunchBlock** is preserved.

Interception: Usually intercepted by any object on the MGCNLT_APP_STARTUP list.

■ MSG_META_UPDATE_WINDOW

```
void        MSG_META_UPDATE_WINDOW(
            UpdateWindowFlags   updateFlags,
            VisUpdateMode       updateMode);
```

This message is sent as part of the system's window update mechanism. Typically, this message is sent to windowed objects on the GAGNLT_WINDOWS list when the GenApplication object becomes GS_USABLE.

1.1

The message passes a bitfield of **UpdateWindowFlags**. These flags determine the type of action prompting the window update.

UWF_ATTACHING
If set, the message is being sent because the application is attaching.

UWF_DETACHING
If set, the message is being sent because the application is detaching.

UWF_RESTORING_FROM_STATE
If set, the application is restoring from state. This flag will only be set if UWF_ATTACHING is also set.

UWF_FROM_WINDOWS_LIST
If set, the message is being sent because the object is on the GAGCNLT_WINDOWS list, and not because it was later built on demand. This flag will only be set if UWF_ATTACHING is also set.

Source: Window update mechanism.

Destination: Entries on the Application's GAGCNLT_WINDOWS list.

Parameters: *updateFlags* **UpdateWindowFlags**.

updateMode **VisUpdateMode**.

Interception: Generally not intercepted.

Objects

■ MSG_META_DETACH

```
void        MSG_META_DETACH(
            word   callerID,
            optr   caller);
```

This message severs the links between an object and the rest of the system. The exact way this is handled depends on the object being detached. For full information on detaching objects, see “GEOS Programming,” Chapter 5 of the Concepts Book.

The “state” of the object is left intact, in case an image of the object needs to be saved away in a state file for later recreation. MSG_META_DETACH sent to an application’s process will start the process by which it is detached from the system, and then exited. MSG_META_DETACH is asynchronous, in that it need not complete its job immediately upon being called. Rather, it may take as much time, invoking and waiting for the completion of subsidiary detaches (say of child objects needing to perform special actions to detach, or of threads created earlier), before it responds with MSG_META_ACK to let its caller know that the detach has completed.

Source: Kernel, other objects relaying detach message.

Destination: GenProcess, GenApplication, objects on active lists.

Parameters:

callerID	Object which sent message.
caller	Object which should be sent a MSG_META_ACK when detaching object has finished.

Return: Nothing.

Interception: Intercepted as a means of finding out that the application is shutting down. Call the superclass in case it needs such notification.
If you create additional threads, or object classes which need to be notified when the application is about to be exited, you may need to extend the detach mechanism by intercepting MSG_META_DETACH in a subclass of an object already receiving that message, such as GenApplication, GenControl, GenInteraction dialogs, etc. You must make sure that all objects you’ve sent MSG_META_DETACH to have responded with a MSG_META_ACK before your object can reply with MSG_META_ACK. Remember that your superclass may be sending MSG_META_DETACH. The kernel provides some default behavior in **MetaClass**, and some utility routines, to make this a simpler task. The default handler for MSG_META_DETACH, for instance, at a leaf object

(one which doesn't propagate the MSG_META_DETACH), performs the required response (sending a MSG_META_ACK). Thus, leaf objects can just intercept MSG_META_DETACH for notification purposes, then call the superclass, and worry no more. The utility routines **ObjInitDetach()** and **ObjEnableDetach()** work in conjunction with a default MSG_META_ACK handler in **MetaClass** to keep track of how many outstanding acknowledgments are being waited for, and call MSG_META_DETACH_COMPLETE on your object once all acknowledgments have returned (the count reaches zero). The default handler for MSG_META_DETACH_COMPLETE then generates the acknowledgment response required of your object to complete its detach. You may optionally call the superclass before sending the detach message to your children and dependents, depending on which order you want things to detach in. The call to the superclass must happen between the **ObjInitDetach()** and **ObjEnableDetach()**, however.

■ MSG_META_DETACH_COMPLETE

```
void        MSG_META_DETACH_COMPLETE();
```

This message is sent to an object being detached when all of its children and active participants have acknowledged the detach. For full information on detaching objects, see "GEOS Programming," Chapter 5 of the Concepts Book.

MSG_META_DETACH_COMPLETE is sent to the object which called **ObjInitDetach()**. This will happen when as many acknowledgments have been received as **ObjIncDetach()** was called, and **ObjEnableDetach()** was called. The **MetaClass** handler for this message sends MSG_META_ACK to the OD passed to the **ObjInitDetach()** call. This message is provided so that an object will know when all of its children have detached. Note that this message is received only if **ObjInitDetach()** has been called for this object. Note also that your superclass may call **ObjInitDetach()** without your knowing.

Source: **MetaClass** handler for MSG_META_ACK, if detach count has dropped to zero (i.e. no outstanding requests), for objects that are detach nodes only (make use of **ObjInitDetach()** or **ObjEnableDetach()**).

Destination: Self.

Parameters: None.

Return: Nothing.

Interception: This is a handy message to intercept when using the **ObjInitDetach()** mechanism and need to know when all objects asked to detach have responded. Calling the superclass at this point in time will cause an MSG_META_ACK to go back to whatever object sent the MSG_META_DETACH to this object originally. There is no requirement to call the superclass at this time, and in fact this is a way to prolong the detach cycle for this object— by simply starting up another **ObjInitDetach()** sequence, for instance.

■ MSG_META_DETACH_ABORT

```
void        MSG_META_DETACH_ABORT();
```

This message causes a detach to be aborted. This can cause some very complex synchronization problems and should not be used lightly. You will find very little call to use it.

Source: Renegade object on active list, after having received MSG_META_DETACH, as an alternative to replying with MSG_META_ACK.

Destination: The optr passed in MSG_META_DETACH.

Interception: Handled by GenField to deal with applications that refuse to die, and GenSystem for Fields that have problem applications. Other than that, any detach node wishing to provide this service will have to figure out a way to do it itself.

Parameters: None.

Return: Nothing.

■ MSG_META_APP_SHUTDOWN

```
void        MSG_META_APP_SHUTDOWN(
            word                callerID,
            optr                ackOD);
```

This message is the complement to MSG_META_APP_STARTUP. This message is sent to objects on the MGCNLT_APP_STARTUP list before an application exits but after the UI for the application is detached. Essentially, it operates in the same manner as MSG_META_DETACH except that the receiving object sends MSG_META_SHUTDOWN_ACK when its shutdown is complete.

Source: Sent by **GenProcessClass** after detaching the UI but before exiting the application; if the UI was never attached (i.e. it handled MSG_META_APP_STARTUP but not MSG_META_ATTACH) the UI will obviously not be detached.

Destination: Any object that needs to be notified when the application is about to exit.

Parameters: *callerID* Word of data for caller's use.

ackOD Optr of object to be sent MSG_META_SHUTDOWN_ACK.

Interception: Usually intercepted by objects on the MGCNLT_APP_STARTUP list.

■ MSG_META_SHUTDOWN_COMPLETE

```
void        MSG_META_SHUTDOWN_COMPLETE();
```

This message is sent to the object that initiated the detach sequence after it has received MSG_META_SHUTDOWN_ACK for each **ObjIncDetach()** that was previously called. This message is only sent if **ObjInitDetach()** was previously called passing the message MSG_META_APP_SHUTDOWN.

The default handler for this message sends MSG_META_SHUTDOWN_ACK to the object passed in the original **ObjInitDetach()** call.

Source: **MetaClass** handler for MSG_META_SHUTDOWN_ACK if detach count reaches zero (i.e. no outstanding requests), for objects that are shutdown nodes only (i.e. make use of **ObjInitDetach()**.)

Destination: The object sends this message to itself.

Interception: Intercept if you are using the **ObjInitDetach()** mechanism and need to be notified when all objects have been notified of the detach.

■ MSG_META_SHUTDOWN_ACK

```
void        MSG_META_SHUTDOWN_ACK(
            word                callerID,
            optr                ackOD);
```

This message is sent back in response to a MSG_META_APP_SHUTDOWN. This message serves to notify the object the object has fulfilled the request.

Source: Object having received MSG_META_APP_SHUTDOWN. The default handler will dispatch MSG_META_SHUTDOWN_ACK after **MetaClass** has processed MSG_META_APP_SHUTDOWN. (You could, of course, intercept MSG_META_APP_SHUTDOWN and send MSG_META_SHUTDOWN_ACK yourself in your handler.)

Destination: Optr passed in MSG_META_APP_SHUTDOWN.

Parameters:		
	callerID	Data passed in MSG_META_APP_SHUTDOWN.
	ackOD	Object which has completed shutting down.

Interception: **MetaClass** provides default handling for this message when using the **ObjInitDetach()** mechanism. Objects not using this mechanism will want to intercept this message if there is a need to know when the object has completed shutting down.

■ MSG_META_ACK

```
void        MSG_META_ACK(
            word   callerID,
            optr   caller);
```

This message acknowledges a detach message. It is sent by objects that have been notified of another object's detach. The default handler for MSG_META_DETACH simply sends MSG_META_ACK back to the object that sent the detach message.

Source: Object having received MSG_META_DETACH (default handler in **MetaClass** will reflexively respond to any MSG_META_DETACH with a MSG_META_ACK, though you can change this behavior either by using **ObjInitDetach()** or by not letting the message get to the **MetaClass** handler, and responding yourself with a MSG_META_ACK sometime later).

Destination: The optr passed in MSG_META_DETACH.

Interception: **MetaClass** provides default handling of this message, for objects using the **ObjInitDetach()** mechanism. Objects *not* using this mechanism will want to intercept this if there is a need to know when the object asked to detach earlier has completed its detach.
MSG_META_ACK is normally inherited from **MetaClass** which calls **ObjEnableDetach()**. This routine decrements the detach count, and when that count reaches zero, sends a MSG_DETACH_COMPLETED to the object itself.

Warnings: If you are expecting a MSG_META_ACK back from anything, make sure you are using the mechanism initiated with **ObjInitDetach()** yourself, or you should handle MSG_META_ACK to prevent **MetaClass** from assuming you *are* using such a mechanism.

Parameters:	*callerID*	data passed to MSG_META_ACK
	caller	object which has completed detaching

Return: Nothing.

■ MSG_META_BLOCK_FREE

1.1 **void** MSG_META_BLOCK_FREE();

This message initiates a sequence which will free an entire object block when received by any object within that block. The block will be freed when its in-use count reaches zero and the message queues for the block have been cleared.

This is a fairly low-level operation, and should be performed only after the objects in the block have been removed from any tree(s) they are attached to, and are otherwise "shut down." For generic objects, this generally means first calling MSG_GEN_SET_NOT_USABLE, then MSG_GEN_REMOVE_CHILD before using this message. For Visible objects, MSG_VIS_REMOVE will both visually shut down the visible tree, and then remove it from its parent.

Source: Unrestricted.

Destination: Any object within a block that is ready to have a low-level delete performed on it (i.e. isn't on screen, isn't linked to objects in other blocks, etc.).

Interception: Unnecessary, as **MetaClass** does the right thing.

Parameters: None.

Return: Nothing.

■ MSG_META_OBJ_FREE

void MSG_META_OBJ_FREE();

This message initiates a sequence which will free an object. The object will be freed after its message queues have been flushed.

This is a fairly low-level operation, and should be performed only after the object has been removed from any tree it is attached to and is otherwise "shut down." Consider using MSG_GEN_DESTROY for generic objects, MSG_VIS_DESTROY for visible ones.

◆ Objects

Source: Unrestricted.

Destination: Any object within a block that is ready to have a low-level delete performed on it (i.e. isn't on screen, isn't linked to objects in other blocks, etc.).

Interception: Unnecessary, as **MetaClass** does the right thing.

Parameters: None.

Return: Nothing.

■ MSG_META_DEC_BLOCK_REF_COUNT

```
void      MSG_META_DEC_BLOCK_REF_COUNT(
          MemHandle           block1,
          MemHandle           block2);
```

This message is a utility message to call **MemDecRefCount()** on one or two memory handles.

This message is useful for IACP, which initializes the reference count to the number of servers returned by **IACPConnect()** and records this message as the message to be returned. After each server has processed its information, the reference count will return to zero and the handles will be freed.

Source: Unrestricted.

Destination: Any object.

Parameters: *block1* Handle of a block whose reference count should be decremented, or 0 if none.

Parameters: *block2* Handle of a block whose reference count should be decremented, or 0 if none.

Interception: Generally not intercepted.

■ MSG_META_OBJ_FLUSH_INPUT_QUEUE

```
void      MSG_META_OBJ_FLUSH_INPUT_QUEUE(
          EventHandle event,
          ObjFlushInputQueueNextStop nextStop,
          MemHandle objBlock);
```

This message clears out the message queues associated with an object. This is rarely, if ever called from within an application, and there is little call to subclass it.

This queue-flushing mechanism is used in the Window, Object, and Object Block death mechanisms. Objects that implement their own “hold up input” queues must redirect this message through that queue, so that it is flushed as well.

Source: Kernel (**WinClose()**, **WinSetInfo()**, **ObjFreeObjBlock()**, MSG_META_OBJ_FREE, or MSG_META_BLOCK_FREE).

Destination: Should first be sent to the Kernel’s Input Manager (See the routine **ImInfoInputProcess()**). The message is then relayed first to the System Input Object (usually the **GenSystemClass** object), then to the Geode Input Object (usually a **GenApplicationClass** object), and finally to the owning process, which dispatches the passed event.

Parameters: *event* Event to dispatch upon conclusion of flush.

objBlock Block Handle that flushing is being performed for (generally the handle of the destination object in the above event). This is the block from which the “OWNING GEODE”, as referenced in the **ObjFlushInputQueueNextStop** enumerated type, is determined.

nextStop **ObjFlushInputQueueNextStop** (Zero should be passed in call to first object, from there is sequenced by default **MetaClass** handler)

Return: Nothing.

Structures:

```
typedef enum {
      OFIQNS_SYSTEM_INPUT_OBJ,
      OFIQNS_INPUT_OBJ_OF_OWNING_GEODE,
      OFIQNS_PROCESS_OF_OWNING_GEODE,
      OFIQNS_DISPATCH
} ObjFlushInputQueueNextStop;
```

Interception: Default **MetaClass** handler implements relay of message from one object to the next, and dispatches the passed event. *Must* be intercepted by any input-flow controlling objects (System object, VisContent) which implement “hold-up” queues that hold up input-related messages. The handlers in such cases should pipe this method through the hold up queue as it does with the other messages, and finish up when it comes out by sending this message, with all data intact, to the superclass for continued default processing.

1.1.2.2 Class Messages

These messages are utilities that identify the class of a particular object. You should not subclass these. Their use is shown in “GEOS Programming,” Chapter 5 of the Concepts Book.

■ MSG_META_GET_CLASS

```
ClassStruct * MSG_META_GET_CLASS();
```

This message returns a pointer to the **ClassStruct** structure of the recipient object’s class.

Source: Unrestricted.

Destination: Any object.

Parameters: None.

Return: The object’s class.

Interception: Don’t.

■ MSG_META_IS_OBJECT_IN_CLASS

```
Boolean   MSG_META_IS_OBJECT_IN_CLASS(
          ClassStruct * class);
```

This message determines whether the recipient object is a member of a given class (or a subclass of the given class). If the return is *true*, the object is in the class. If *false*, the object is not in the class. If a variant class is encountered (when checking to see if the object is an instance of a subclass of the passed class), the object will not be grown out past that class in the search. If you want to do a complete search past variant classes, send a MSG_META_DUMMY first.

Source: Unrestricted.

Destination: Any object.

Parameters: *class* Class to see if object is a member of.

Return: Returns *true* if object is a member of the passed class (or a subclass), *false* otherwise.

Interception: Don’t.

1.1.2.3 Object Management Messages

These messages fill in and resolve an object's instance data. They should usually not be subclassed, and will be sent by applications infrequently (if ever).

■ MSG_META_RESOLVE_VARIANT_SUPERCLASS

```
ClassStruct * MSG_META_RESOLVE_VARIANT_SUPERCLASS(
                word   MasterOffset);
```

1.1

This message is sent by the object system when it needs to know the run-time superclass of a particular object's variant master class. The system sends this message to the object when it first attempts to deliver a message to the superclass of a variant class. The object must examine itself and determine what its superclass for that master level should be.

Source: Object system.

Destination: Any object with a variant class in its class hierarchy.

Interception: Because variant master classes tend to be strictly administrative in nature, providing useful and very generic functionality to their subclasses, all immediate children of a variant master class will need to intercept this message and return the appropriate class pointer.

Parameters: *MasterOffset* Master offset of the level being resolved. If you know there's a variant class above your own, you will need to examine this to determine if it is your master level whose variant is being resolved, or the one above you.

Return: Superclass to use.

■ MSG_META_RELOCATE

```
Boolean  MSG_META_RELOCATE(
         word                vMRelocType,
         word                frame);
```

This message is sent by the object system to evaluate and resolve all of the object's relocatable instance data fields (pointers, optrs, etc.). Note that this only applies if the class' CLASSF_HAS_RELOC flag is set.

NOTE: The calling of this method is non-standard in that it does not pass through the class's method table. Rather, the handler address is placed after the method table and a direct call is issued. This means a relocation routine should not be bound to MSG_META_RELOCATE but should rather be bound to **@reloc**, which Goc understands to mean the handler is for both MSG_META_RELOCATE and MSG_META_UNRELOCATE.

Note also that relocation-by-routine happens in addition to (but before) any relocation due to the class' relocation table. To suppress relocation-by-table, you should initialize the class record with the CLASSF_HAS_RELOC flag to prevent Goc from generating a table for the class.

Source: Kernel, when loading in object block, general resources, or object blocks stored in VM file format.

Destination: Individual object needing relocations beyond what the kernel can do automatically (or that simply request for this message to be sent by having their CLASSF_HAS_RELOC bit set)

Interception: Intercepted by any class needing to perform special relocations on its instance data. Superclass should be called, in case a superclass also needs to perform this operation on its own instance data.

Parameters: *vmRelocType* Type giving some context to the relocation.

frame Frame to pass to **ObjRelocOrUnRelocSuper()**.

Return: If an error occurred, this will return *true*.

Structures:

```
typedef enum {
      VMRT_UNRELOCATE_BEFORE_WRITE,
      VMRT_RELOCATE_AFTER_READ,
      VMRT_RELOCATE_AFTER_WRITE,
      VMRT_RELOCATE_FROM_RESOURCE,
      VMRT_UNRELOCATE_FROM_RESOURCE
} VMRelocType;
```

Warnings: This method may not call **LMemAlloc()**, **LMemReAlloc()**, or **LMemFree()**.

■ MSG_META_UNRELOCATE

```
Boolean MSG_META_UNRELOCATE(
        word                vMRelocType,
        word                frame);
```

This message causes an object to unresolve all its relocatable instance data fields, returning them to special index values.

Source: Kernel, when loading in object block, general resources, or object blocks stored in VM file format.

Destination: Individual object needing relocations beyond what the kernel can do automatically (or that simply request for this message to be sent by having their CLASSF_HAS_RELOC bit set).

Interception: Intercepted by any class needing to perform special relocations on its instance data. Superclass should be called, in case a superclass also needs to perform this operation on its own instance data.

Parameters:

vmRelocType	Type giving some context to the relocation.
frame	Frame to pass to **ObjRelocOrUnRelocSuper()**.

Return: If an error occurred, this will return *true*.

Structures:

```
typedef enum {
      VMRT_UNRELOCATE_BEFORE_WRITE,
      VMRT_RELOCATE_AFTER_READ,
      VMRT_RELOCATE_AFTER_WRITE,
      VMRT_RELOCATE_FROM_RESOURCE,
      VMRT_UNRELOCATE_FROM_RESOURCE
} VMRelocType;
```

Warnings: This method may not call **LMemAlloc()**, **LMemReAlloc()**, or **LMemFree()**.

1.1.2.4 User Interface Utility Meta Messages

These messages are used primarily by the User Interface. You will have very little call to subclass or send them.

■ MSG_META_SET_FLAGS

```
void        MSG_META_SET_FLAGS(
            ChunkHandle objChunk,
            ObjChunkFlags bitsToSet,
            ObjChunkFlags bitsToClear);
```

This message sets the chunk flags for an object. The chunk flags determine how the object is handled with regard to state saving, dirty state, etc.

Source: Unrestricted.

Destination: Any object.

Interception: Unnecessary, as **MetaClass** does the right thing.

Parameters:

objChunk	chunk to set flags for.
bitsToSet	bits to set.
bitsToClear	bits to clear.

Return: Nothing.

■ MSG_META_GET_FLAGS

```
word        MSG_META_GET_FLAGS( /* low byte = ObjChunkFlags */
            ChunkHandle ch);
```

This message returns the chunk flags for the object. This works just like the **ObjGetFlags()** routine, but can be used when the object queried is being run by a different thread.

Source: Unrestricted.

Destination: Any object.

Interception: Unnecessary, as **MetaClass** does the right thing.

Parameters:

objChunk	chunk to get flags for.

Return: Word with **ObjChunkFlags** in low byte, zero in high byte.

■ MSG_META_QUIT

```
void        MSG_META_QUIT();
```

This message, when sent to a GenApplication object, initiates the shutdown sequence for the application. All affected objects are notified.

GenApplication does some error checking for multiple quits or detaches and then starts this sequence by passing MSG_META_QUIT(QL_BEFORE_UI) to the process. The default process handler for MSG_META_QUIT varies depending on the **QuitLevel**, which is passed in, but only when sent to the process (see MSG_META_QUIT_PROCESS alias, below).

The method handler for each level of quit should then send MSG_META_QUIT_ACK with the same **QuitLevel** when it is done. The default behavior for a process' MSG_META_QUIT responses are:

QL_BEFORE_UI
Sends MSG_META_QUIT_ACK to self via queue.

1.1

QL_UI Sends MSG_GEN_APPLICATION_INITIATE_UI_QUIT(0) to the GenApplication.

QL_AFTER_UI
Sends MSG_META_QUIT_ACK to self via queue.

QL_DETACH Sends MSG_META_DETACH to self via queue.

QL_AFTER_DETACH
Sends MSG_META_QUIT_ACK to self via queue.

The generic UI objects are first asked to quit via MSG_GEN_APPLICATION_INITIATE_UI_QUIT when sent to a GenApplication (active list). It will cause MSG_META_QUIT to be sent to all objects on the active list that are marked as desiring them. These objects on the active list can handle the MSG_META_QUIT any way they please. The process will be notified by a MSG_META_QUIT_ACK with the **QuitLevel** set to QL_UI.

Source: Unrestricted.

Destination: GenApplication object (note that this message has aliases so that it may be sent to a Process object, or any object).

Parameters: None.

Return: Nothing.

Interception: Unlikely.

■ MSG_META_QUIT_PROCESS

```
@alias (MSG_META_QUIT)
 void    MSG_META_QUIT_PROCESS(
         word                quitLevel,
         ChunkHandle         ackODChunk);
```

For information about the quit mechanism, see MSG_META_QUIT, above.

◆Objects

The process's MSG_META_QUIT_ACK handler is what causes this walking down the **QuitList**; It provides the following behavior for each **QuitLevel**:

QL_BEFORE_UI
Sends MSG_META_QUIT(QL_UI) to self.

QL_UI Sends MSG_META_QUIT(QL_AFTER_UI) to self.

QL_AFTER_UI
Sends MSG_META_QUIT(QL_DETACH) to self.

QL_DETACH Sends MSG_META_QUIT(QL_AFTER_DETACH) to self.

QL_AFTER_DETACH
Sends MSG_GEN_PROCESS_FINISH_DETACH to self.

Source: Unrestricted.

Destination: Process object.

Parameters: *quitLevel* What stage of quitting we are in.

ackODChunk Acknowledgment OD to be passed on to MSG_META_QUIT_ACK.

Return: Nothing.

Interception: Unlikely.

Warnings: You cannot abort the quit at the QL_DETACH stage or later.

■ MSG_META_QUIT_OBJECT

```
@alias (MSG_META_QUIT)
 void     MSG_META_QUIT_OBJECT(
          optr   obj);
```

For information about the quit mechanism, see MSG_META_QUIT, above.

Source: Unrestricted.

Destination: Process object.

Parameters: *obj* Object to send MSG_META_QUIT_ACK to.

Return: Nothing.

Interception: Unlikely.

MSG_META_QUIT_ACK

```
void      MSG_META_QUIT_ACK(
          word                  quitLevel,
          word                  abortFlag);
```

This message is sent to a Process object in response to a MSG_META_QUIT. The Process object handles this message by continuing the quit sequence.

Source: Any object having received MSG_META_QUIT

Destination: OD passed in MSG_META_QUIT.

Parameters: *quitLevel* **QuitLevel** acknowledging (if responding to a process).

abortFlag (non-zero if you want to abort the quit).

Return: Nothing.

Warnings: For processes that subclass MSG_META_QUIT, you cannot abort the quit at the QL_DETACH stage or later.

MSG_META_FINISH_QUIT

```
void      MSG_META_FINISH_QUIT(
          Boolean               abortFlag);
```

This message is sent to the object that initiated MSG_META_QUIT and has received MSG_META_QUIT_ACK from each party notified. This message informs the object that it has finished sending out all MSG_META_QUIT messages and can go on with quitting (or aborting the quit if that is the case).

Source: Object that initiated MSG_META_QUIT.

Destination: Any object.

Parameters: *abortFlag* (non-zero if you want to abort the quit).

1.1.2.5 Event Messages

These messages are used to send classed events to other objects. A classed event is typically an event stored earlier with the Goc keyword **@record**.

Objects

■ MSG_META_DISPATCH_EVENT

```
Boolean  MSG_META_DISPATCH_EVENT(
         AsmPassReturn *retVals,
         EventHandle eventHandle,
         MessageFlags msgFlags););
```

This message causes an object to **@send** or **@call** a message of another object. This is useful for getting one object run by a different thread to call yet another object or to send a reply to the first object. This message can cause complex synchronization problems if not used with extreme care. 1.1

Source: Unrestricted.

Destination: Any object.

Interception: Unnecessary, as **MetaClass** does the right thing.

Parameters:

retValue	structure to hold return values.
eventHandle	Event which will be sent.
msgFlags	flags which will determine how message is sent.

Return: If MF_CALL specified, then carry flag return value will be returned.

Structures:

```
typedef struct {
      word  ax;
      word  cx;
      word  dx;
      word  bp;
} AsmPassReturn;
```

■ MSG_META_SEND_CLASSED_EVENT

```
void     MSG_META_SEND_CLASSED_EVENT(
         EventHandle         event,
         TravelOption        whereTo);
```

This message is similar to several MSG_GEN_SEND_... messages defined in **GenClass**. This message sends a previously recorded classed event to a certain type of destination defined in the **TravelOption** argument *whereTo*.

This message's interesting behavior is actually added by the User Interface, which defines **TravelOption** types. See the message definition in **GenClass** for details. The default behavior provided here in **MetaClass** is to destroy the event if TO_NULL is passed, else to deliver the event to itself if it is capable of handling it (the object is a member of the class stored with the event). The event is always freed, whether or not it is deliverable.

MetaClass recognizes the following **TravelOption** values:

TO_NULL
TO_SELF
TO_OBJ_BLOCK_OUTPUT
TO_PROCESS

1.1

TO_OBJ_BLOCK_OUTPUT sends the event to the object block's output set in its block header.

Source: Unrestricted.

Destination: Any object.

Interception: By default, **MetaClass** handlers deal with just the most primitive of the **TravelOption** values. Object classes can add new **TravelOption** types, but must then intercept this message to implement them (calling the superclass if it doesn't recognize the **TravelOption** passed).

Parameters: *event* Classed event, probably created using **@record**.

whereTo **TravelOption** describing target of message.

Return: Nothing.

■ MSG_META_GET_OBJ_BLOCK_OUTPUT

```
optr    MSG_META_GET_OBJ_BLOCK_OUTPUT();
```

This message returns the output optr of an object block that contains the object sent the message.

Source: Unrestricted.

Destination: Any object (except a process object).

Return: Optr of the block's output field.

Interception: Generally not intercepted.

■ MSG_META_SET_OBJ_BLOCK_OUTPUT

```
void    MSG_META_SET_OBJ_BLOCK_OUTPUT(
        optr                output);
```

This message sets the object block output—the block containing the object sent the message— to the passed optr.

◆ Objects

Source: Unrestricted.

Destination: Any object (except a process object).

Parameters: *output* Optr of the object to act as the block's output.

Interception: Generally not intercepted.

■ MSG_META_GET_OPTR

```
optr    MSG_META_GET_OPTR();
```

This message returns the object's optr. This is useful when combined with MSG_GEN_GUP_CALL_OBJECT_OF_CLASS to get the optr of an object of a given class somewhere up in a Generic Tree.

Note: MSG_GEN_GUP_CALL_OBJECT_OF_CLASS dies if an object of the class doesn't exist. Use MSG_GEN_GUP_TEST_FOR_OBJECT_OF_CLASS before using MSG_GEN_GUP_CALL_OBJECT_OF_CLASS if there is some question as to whether an object of a given class exists.

Source: Unrestricted.

Destination: Any object.

Interception: Unlikely.

Parameters: None.

Return: The object's optr.

■ MSG_META_GET_TARGET_AT_TARGET_LEVEL

```
void    MSG_META_GET_TARGET_AT_TARGET_LEVEL(
        GetTargetParams *retValue,
        TargetLevel level);
```

This message returns the **GetTargetParams** structure containing, among other things, the current target object at a given target level. The **MetaClass** handler simply returns information about the current object since it is assumed to be the current target. See "Input," Chapter 11 of the Concepts Book, for information on target.

Source: Unrestricted.

Destination: Any object.

Interception: Must be handled by target nodes to correctly pass the request on down to the next target below current node in hierarchy.

Parameters: *level* Zero for leaf, otherwise **TargetLevel**, as defined by UI.

retValue Structure to hold return value.

Return: Nothing returned explicitly.

retValue Filled with return values.

Structures:

```
typedef struct {
      ClassStruct              *GTP_class;
      optr                     GTP_target;
} GetTargetParams;
```

1.1.2.6 Variable Data Messages

Variable data is instance data that can appear or not appear within the object's instance chunk. For information on variable data and how these three messages are used, see "GEOS Programming," Chapter 5 of the Concepts Book.

■ MSG_META_ADD_VAR_DATA

```
void      MSG_META_ADD_VAR_DATA(@stack
          word   dataType,
          word   dataSize,
          word   *data)
```

This message adds a variable data type to the recipient object's instance data. If the variable data field was already present, this will change its value. This is useful for adding hints to generic objects at run-time.

Note that the object will be marked dirty even if nothing was changed.

NOTE: The *dataType* should have VDF_SAVE_TO_STATE set as desired. VDF_EXTRA_DATA is ignored; it will be set correctly by this routine.

Source: Unrestricted.

Destination: Any object.

Interception: Generally not intercepted; default **MetaClass** handling performs the desired function.

Parameters: *dataType* Data type (e.g. ATTR_PRINT_CONTROL_APP_UI).

dataSize Size of data, if any.

data If no extra data, NULL. If *dataSize* is non-zero, then this may be a pointer to data to initialize data with.

Return: Nothing. Object marked dirty even if data type already exists.

■ MSG_META_DELETE_VAR_DATA

```
Boolean   MSG_META_DELETE_VAR_DATA(
          word   dataType);
```

This message removes a particular variable data entry from the recipient object's instance data. This is useful for removing hints from generic objects at run-time.

1.1

Source: Unrestricted.

Destination: Any object.

Interception: Generally not intercepted; default **MetaClass** handling performs the desired function.

Parameters: *dataType* Data type to delete. **VarDataFlags** ignored.

Return: Returns *false* if data deleted, *true* if data was not found. Object marked dirty if data type found and deleted.

■ MSG_META_INITIALIZE_VAR_DATA

```
word      MSG_META_INITIALIZE_VAR_DATA(
          word   dataType);
```

This message is sent to an object any time the **ObjVarDerefData()** routine is called and the data type is not found. It should be subclassed by any object that defines a variable type that will be used with **ObjVarDerefData()**. The object must create and initialize the data and return its offset.

Sent to an object having a variable data entry which code somewhere is attempting to access via **ObjVarDerefData()**. It is the object that defines the variable data entry type's responsibility to create the data entry and initialize it at this time, and to return a pointer to the extra data (if any), as returned by **ObjVarAddData()**.

Source: **ObjVarDerefData()** routine. Should not be used as a replacement for **ObjVarAddData()**, or MSG_ADD_VAR_DATA_ENTRY, but may be used any time code is ready to access a particular piece of variable data instance data, knows that the variable data has not yet been created, and wishes to ensure that it does exist.

Destination: Any object stored in an LMem Object block.

Interception: Required by any class which defines a variable data entry type that needs to be initialized before usage. Objects handling this message should first compare the passed data type against variable data types it understands, and pass any unknown types onto the superclass for handling.

Parameters: Variable data type.

Return: Offset to extra data created (or, if no extra data, the start of data entry plus the size of **VarDataEntry**). Normally, this would just be the offset returned by the call to **ObjVarAddData()**.

■ MSG_META_GET_VAR_DATA

```
word MSG_META_GET_VAR_DATA( /* returns size of data returned in buf;
                             * -1 if not found */
         word   dataType,
         word   bufSize,
         void   *buf);
```

This message fetches variable data of a given type from an object.

Source: Unrestricted.

Destination: Any object.

Interception: Generally not intercepted; default **MetaClass** handling performs the desired function.

Parameters:

dataType	The variable data category to return.
bufSize	Size available to return data.
buf	Pointer to buffer to hold returned data.

Return: The size of the data returned. If the vardata entry was not found, then message will return -1.

buf	Filled with vardata's data, if any.

1.1.2.7 Notification Messages

These messages are used by the various notification mechanisms throughout the system.

Objects

■ MSG_META_NOTIFY

```
void        MSG_META_NOTIFY(
            ManufacturerID manufID,
            word   notificationType,
            word   data);
```

This message notifies the recipient that some change or action has taken place. The object must have registered for the notification. The type of change that has occurred depends on the *notificationType* argument.

One word of notification data is allowed, but this should not reference a handle which must at some point be destroyed. See MSG_META_NOTIFY_WITH_DATA_BLOCK for such requirements.

Source: Unrestricted.

Destination: Any object, or any of the **GCNListSend()** routines.

Interception: No general requirements, though particular notification types may place restrictions or requirements on such handling.

Parameters:

manufID	Manufacturer ID associated with notification type.
notificationType	What sort of notification is being announced.
data	One word of data, which will be placed in a vardata field. If more than one word is necessary, use MSG_META_NOTIFY_WITH_DATA_BLOCK, below.

Return: Nothing.

■ MSG_META_NOTIFY_WITH_DATA_BLOCK

```
void        MSG_META_NOTIFY_WITH_DATA_BLOCK(
            ManufacturerID manufID,
            word   notificationType,
            MemHandle data);
```

This message acts like MSG_META_NOTIFY, but it also carries a handle of a block of data. It is absolutely imperative that if this message is subclassed, the object call its superclass in the handler.

The data block must be set up to use the Block “reference count” mechanism, i.e. be sharable and initialized with **MemInitRefCount()**. Details on the count are noted below.

Source: Unrestricted.

Destination: Any object, or any of the GCNListSend... routines

Interception: Message *must* eventually arrive at the **MetaClass** handler, with the handle to the data block with the reference count intact, in order for the block to be freed when no longer referenced. Failure to do so will result in garbage being left on the heap, which will kill the system with repetitive occurrences.

Parameters:

manufID	Manufacturer ID associated with notification type.
notificationType	What sort of notification is being announced.
data	SHARABLE data block having a "reference count" initialized via **MemInitRefCount()**.

1.1

NOTE on data block reference counts:
The reference count should hold the total number of references of this data block. This count should be incremented before sending a message holding a reference to this block (using **MemIncRefCount()**). Any messages passing such reference either must have a **MetaClass** handler which decrements this count and frees the block if it reaches zero or must call **MemDecRefCount()** (which does exactly that). **GCNListSend()** and similar functions add in the number of optrs in any list to which a message referring to this block is sent. Thus, when creating a block which will only be sent using **GCNListSend()**, this count should be initialized to zero. If the block is to be sent to one or more objects or **GCNListSend()** calls, the calling routine should call **MemIncRefCount()** before making the calls, being sure to call **MemIncRefCount()** additionally for any objects that the message is sent to, and then call **MemDecRefCount ()** after the calls, to balance the increment call at the start.

Return: Nothing.

Warnings: This message must eventually reach the default **MetaClass** handler, so that the block can be freed when no longer referenced.

■ MSG_META_GCN_LIST_ADD

```
Boolean   MSG_META_GCN_LIST_ADD(@stack
          optr                   dest,
          word                listType,
          ManufacturerID      listManuf);
```

This message adds the passed object to a particular notification list. It returns *true* if the object was successfully added, *false* otherwise. This message is the equivalent of **GCNListAdd()** for individual object GCN list.

Source: Unrestricted.

Destination: Object providing GCN services.

Interception: Unnecessary, as **MetaClass** does the right thing.

Parameters:		
	dest	Object to be added to list.
	listType	**GCNListType** to add object to.
	listManuf	Manufacturer ID associated with GCN list.

Return: Returns *true* if optr added, *false* otherwise.

■ MSG_META_GCN_LIST_REMOVE

```
Boolean    MSG_META_GCN_LIST_REMOVE(@stack
           optr   dest,
           word   listType,
           ManufacturerID listManuf);
```

This message removes the passed object from a particular notification list. It returns *true* if the object was successfully removed, *false* otherwise. This message is the equivalent of **GCNListRemove()** for an individual object's GCN list.

Source: Unrestricted.

Destination: Object providing GCN services.

Interception: Unnecessary, as **MetaClass** does the right thing.

Parameters:		
	dest	Object to be removed from list.
	listType	Which list to remove object from.
	listManuf	Manufacturer ID associated with GCN list.

Return: Returns *true* if optr found and removed, otherwise returns *false*.

■ MSG_META_GCN_LIST_SEND

```
void       MSG_META_GCN_LIST_SEND(@stack
           GCNListSendFlags    flags,
           EventHandle         event,
           MemHandle           block,
           word                listType,
           ManufacturerID      listManuf);
```

This message sends the given event to all objects in a particular notification list. The event will be freed after being sent. This message is the equivalent of **GCNListSend()** for an individual object GCN list.

Parameters:		
	flags	Flags to pass on to GCNListSend().
	event	Classed event to send to the list.

block Handle of extra data block, if used, else NULL. This block must have a reference count, which may be initialized with **MemInitRefCount()** and incremented for any new usage with **MemIncRefCount()**. Methods in which they are passed are considered such a new usage, and must have **MetaClass** handlers which call **MemDecRefCount()**.

1.1

listType Which GCN list to send event to.

listManuf Manufacturer ID associated with GCN list.

Return: Nothing.

Structures:

```
typedef WordFlags GCNListSendFlags;
/* These flags may be combined using | and &:
      GCNLSF_SET_STATUS,
      GCNLSF_IGNORE_IF_STATUS_TRANSITIONING */
```

GCNLSF_SET_STATUS

Additionally saves the message as the list's current "status." The "status" message is automatically sent to any object adding itself to the list at a later point in time.

GCNLSF_IGNORE_IF_STATUS_TRANSITIONING

Optimization bit used to avoid lull in status when transitioning between two different sources—such as when the source is the current target object, and one has just lost, and another may soon gain, the exclusive. (The bit should be set only when sending the "null," "lost," or "not selected" status, as this is the event that should be discarded if another non-null status comes along shortly). Implementation is *not* provided by the kernel primitive routines, which ignore this bit, but may be provided by objects managing their own GCN lists. GenApplication responds to this bit by delaying the request until after the UI and application queues have been cleared, and then only sets the status as indicated if no other status has been set since the first request. Other objects may use their own logic to implement this optimization as is appropriate. Mechanisms which can not tolerate the delayed status setting nature of this optimization, or require that all changes are registered, should not pass this bit set.

◆Objects

■ MSG_META_GCN_LIST_FIND_ITEM

```
Boolean  MSG_META_GCN_LIST_FIND_ITEM(@stack
         optr                dest,
         word                listType,
         ManufacturerID      listManuf);
```

This message checks whether an object is on a particular GCN list.

Source: Unrestricted.

Destination: Any object providing GCN services.

Parameters: *dest* Optr of object that we are checking.

listType **GCNListType**.

listManuf **ManufacturerID**.

Return: *true* if object is on the GCN list.

Interception: Unnecessary.

■ MSG_META_GCN_LIST_DESTROY

```
void     MSG_META_GCN_LIST_DESTROY();
```

This message completely destroys the GCN setup for the caller. It frees all GCN lists, cached events, and overhead data storage. This should only be used when the object is being freed. You will likely never handle or call this message.

Source: Object providing GCN services, often in handler for MSG_META_FINAL_OBJ_FREE.

Destination: Self.

Interception: Unnecessary, as **MetaClass** does the right thing.

Parameters: Nothing.

Return: Nothing.

■ MSG_META_NOTIFY_OBJ_BLOCK_INTERACTIBLE

```
void     MSG_META_NOTIFY_OBJ_BLOCK_INTERICTABLE(
         MemHandle objBlock);
```

This message is sent to an object block's output object when the block changes from being not in-use to being in-use. An object may handle this message to monitor changes of in-use status.

Source: Kernel.

Destination: Object which is set as the output of an object block resource either by **ObjBlockSetOutput()**, or by being pre-defined in an application resource.

Interception: May be intercepted to learn about change in object block interactable status. No default handling is provided, though you may wish to pass the message onto the superclass in case it is interested in this data as well.

1.1

Parameters: *objBlock* Handle of object block.

Return: Nothing.

■ MSG_META_NOTIFY_OBJ_BLOCK_NOT_INTERACTIBLE

```
void    MSG_META_NOTIFY_OBJ_BLOCK_NOT_INTERACTIBLE(
        MemHandle objBlock);
```

This message is sent to an object block's output object when the block changes from being not in-use to being in-use. An object may handle this message to monitor changes of in-use status.

Source: Kernel.

Destination: Object which is set as the output of an Object Block resource either by **ObjBlockSetOutput()**, or by being pre-defined in an application resource.

Interception: May be intercepted to learn about change in object block interactable status. No default handling is provided, though you may wish to pass the message onto the superclass in case it is interested in this data as well.

Parameters: *objBlock* Handle of object block.

Return: Nothing.

■ MSG_META_VM_FILE_DIRTY

```
void    MSG_META_VM_FILE_DIRTY(
        FileHandle file);
```

This message is sent to all processes that have a VM file open when a block in the file becomes marked dirty for the first time. This is useful if many processes may be sharing a VM file. The VM file must be marked VMA_NOTIFY_DIRTY in its attributes.

Source: Kernel VM code.

◆Objects

Destination: **ProcessClass** object.

Interception: May be intercepted at process to do whatever is desired on this occurrence of this event. Default behavior in **GenProcessClass** sends notification to the current model **GenDocumentGroupClass** object.

Parameters: *file* Handle open to the VM file, from the receiving process's perspective.

Return: Nothing.

1.1.2.8 Options Messages

These messages are used by the User Interface when working with the GEOS.INI files. You will probably never need to subclass or call these messages.

■ MSG_META_SAVE_OPTIONS

```
void        MSG_META_SAVE_OPTIONS();
```

This message saves an object's options to the .INI file for an object. It is sent via the UI's active list mechanism.

Source: Unrestricted.

Destination: GenApplication object, which in turn broadcasts to everything on list of objects having options needing to be saved

Interception: Objects having options to save should intercept this. Superclass should be called in case any of the superclasses needs similar notification.

■ MSG_META_LOAD_OPTIONS

```
void        MSG_META_SAVE_OPTIONS();
```

This message loads the object's setting from the .INI file.

Source: Unrestricted.

Destination: Any object.

Interception: Any object that should load its options should intercept this. Behavior is currently implemented for Generic UI objects.

■ MSG_META_RESET_OPTIONS

```
void        MSG_META_RESET_OPTIONS();
```

This message resets the object's settings from the .INI file to their initial state.

Source: Unrestricted. Sent to all objects on the GAGCNLT_SELF_LOAD_OPTIONS and GAGCNLT_STARTUP_LOAD_OPTIONS lists.

Destination: Any object.

Interception: Any object that wants to reset its options should intercept this. Behavior is currently implemented for Generic UI objects.

■ MSG_META_GET_INI_CATEGORY

```
void        MSG_META_GET_INI_CATEGORY(
            char    *buf);
```

This message returns the .INI file category of the object.

Source: Unrestricted, though generally self.

Destination: Object having options.

Interception: Default handler walks up tree, eventually finding name of application. Can be intercepted at any level to change the category for a branch.

Parameters: *buf* The buffer for .INI category string. This buffer size cannot store more than 64 bytes.

Return: Nothing returned explicitly.

buf String filled with category string.

1.1.2.9 Suspending and Unsuspending

MSG_META_SUSPEND and MSG_META_UNSUSPEND work together to allow objects to optimize recalculation when doing a series of actions. These messages are implemented by various objects in the system (such as the text object and the grobj body). This mechanism is used by **GenControlClass** to optimize recalculation stemming from multiple controller outputs.

An object typically implements these messages by keeping a suspend count and a record of the calculations that were aborted because the object was suspended. When the suspend count reaches zero, the object will perform the calculations.

An object implementing this mechanism should always call its superclass since multiple class levels could be implementing this mechanism.

■ MSG_META_SUSPEND

```
void	MSG_META_SUSPEND();
```

Suspend calculation in an object.

Source: Normally sent by a controller object but can be sent by anything.

Destination: Any object that implements the mechanism described above.

Interception: An object that wants to implement the mechanism described above.

Parameters: None.

Return: Nothing.

■ MSG_META_UNSUSPEND

```
void	MSG_META_UNSUSPEND();
```

Unsuspend calculation in an object.

Source: Normally sent by a controller object but can be sent by anything.

Destination: Any object that implements the mechanism described above.

Interception: An object that wants to implement the mechanism described above.

Parameters: None.

Return: Nothing.

1.1.2.10 Help Files

MSG_META_GET_HELP_FILE, MSG_META_SET_HELP_FILE, MSG_META_BRING_UP_HELP

These help messages are contained within MetaClass to allow help files within any object in the GEOS system.

■ MSG_META_GET_HELP_FILE

```
void        MSG_META_GET_HELP_FILE(
            char                *buf);
```

This message returns the name of the help file attached to the object sent this message. If no help file is found, the default **MetaClass** handler walks up the tree.

Source: Unrestricted.

Destination: Object in object tree containing help.

Parameters: *buf* Pointer to buffer to store the help file name.

Return: Buffer filled in.

Interception: The default handler walks up the tree; you may intercept to change the help file for a branch.

■ MSG_META_SET_HELP_FILE

```
void        MSG_META_SET_HELP_FILE(
            char                *buf);
```

This message sets the help file for the object sent this message.

Source: Unrestricted, though generally an object sends this to itself.

Destination: An object within a tree.

Parameters: *buf* Pointer to help file name. This buffer's size must be at least FILE_LONGNAME_BUFFER_SIZE.

Interception: Generally not intercepted.

■ MSG_META_BRING_UP_HELP

```
void        MSG_META_BRING_UP_HELP();
```

This message finds a help context for the current object tree and sends a notification to bring up help with that context.

Source: Unrestricted, though generally an object sends this to itself.

Destination: An object within a tree.

Interception: The default handler for this message walks up the visual tree (not the generic tree) eventually finding a **GenClass** object with ATTR_GEN_HELP_CONTEXT. You may intercept at any level to change the help context for a branch.

◆Objects

1.1.3 Exported Message Ranges

MetaClass exports a number of ranges of messages for use by its subclasses (such as **GenClass** and **VisClass**) for various purposes. In most cases, you will not need to use any of these ranges for your own messages. The names of these ranges, however, are listed below. For information on exporting and importing message ranges, see "GEOS Programming," Chapter 5 of the Concepts Book.

- MetaWindowMessages
  These messages alert a window's input object and output descriptor to certain important events. For example, MSG_META_EXPOSED announces that part of a window has been exposed and needs to be redrawn.

- MetaInputMessages
  These are very low-level messages, and will only be used by those geodes which wish to circumvent the Input Manager. To work with the Input Manager correctly, see the selection messages in the MetaUIMessages range.

- MetaUIMessages
  These messages may be used by objects which will interact with the UI. While messages for working with the Input Manager and Clipboard are detailed in this chapter, information on other messages may be found in "GenDocument," Chapter 13.

- MetaSpecificUIMessages
  These messages are internal.

- MetaApplicationMessages
  These messages don't have any meaning attached to them; no class defined in the system or any library has a handler for any of these messages. Any object class defined within an application may have a handler for any of these messages.
  This message range was set up so that two or more classes defined within an application could agree on some message numbers.
  Thus, your application could contain the header listed below, and then write handlers for the message in two completely unrelated application-defined classes:

```
 @importMessage MetaApplicationMessages,
      type0 MSG_MYAPP_DO_SOMETHING(
            type1       arg1,
            type2       arg2);
/* … */
@method MyProcessClass, MSG_MYAPP_DO_SOMETHING
/* …Insert Handler here */
/* … */
@method MyDocumentClass, MSG_MYAPP_DO_SOMETHING
/* …Insert Handler here */
```

- MetaGrObjMessages
  This message range is reserved for notification messages associated with the graphic object library.
- MetaPrintMessages
  For information on these messages, see “The Spool Library,” Chapter 17.
- MetaSearchSpellMessages
  These messages are sent out by the SearchReplace and Spell controllers. For information on these messages, see “The Text Objects,” Chapter 10.
- MetaGCNMessages
  There are several system-defined General Change Notification lists which objects may belong to. This message range holds those messages which will be sent to objects on system-defined lists.
- MetaTextMessages
  See “The Text Objects,” Chapter 10.
- MetaStyleMessages
  See “Generic UI Controllers,” Chapter 12.
- MetaColorMessages
  See “Generic UI Controllers,” Chapter 12.
- MetaFloatMessages
  These messages are sent out by the FloatFormat controller. For information on these messages, see “Generic UI Controllers,” Chapter 12.
- MetaSpreadsheetMessages
  See “Spreadsheet Objects,” Chapter 20.

◆Objects

- MetaIACPMessages
  These messages are used to communicate to other objects using the Inter-Application Communication Protocol. IACP is discussed in “Applications and Geodes,” Chapter 6 of the Concepts Book.

1.1.3.1 Window Messages

Because many objects, both Generic UI objects and others, work together to control the behavior of the system windows, a number of messages have been set up in an exported range so that they may be shared among classes. 1.1

Window Update Messages

The following messages are sent to objects responsible for updating views, and if you subclass content objects, you may wish to intercept these messages.

■ MSG_META_EXPOSED

```
@importMessage MetaWindowMessages, void MSG_META_EXPOSED(
          WindowHandle win);
```

This message is sent to a Window’s exposure object any time a portion of the window is visible on screen, has become invalid, and needs to be redrawn. Correct response is to create a GState on the passed window, call **GrBeginUpdate()** with it, redraw the window, and finish by calling **GrEndUpdate()** and freeing the GState. Drawing will be clipped to the invalid area of the window at the time that **GrBeginUpdate()** is called. Invalidations occurring during the redraw will result in the reduction in the size of the update region, and result in another MSG_META_EXPOSED being generated, to repair the new “invalid” area.

Source: Window system.

Destination: Individual window’s exposure object; View’s output descriptor.

Interception: Required, in order for window to be properly updated. Note that **VisContentClass** provides default handler which creates GState, calls **GrBeginUpdate()**, calls MSG_VIS_DRAW on itself, then calls **GrEndUpdate()**.

Parameters: *win* Window handle which may be passed to **GrCreateGState()**.

Return: Nothing.

Messages Sent to Objects Further Up the Input Hierarchy

The following messages are part of the high level windowing mechanism. Most of these messages are passed around at the GenSystem level, and most object classes defined by applications will not intercept them. Instead, system objects will intercept these messages and pass appropriate messages on to application objects.

1.1

■ MSG_META_WIN_CHANGE

```
@importMessage MetaWindowMessages, void MSG_META_WIN_CHANGE();
```

Sent to the System Input Object (Normally the UI's GenSystem obj), when the pointer position, as passed to the window system in calls to **WinMovePtr()**, has possibly moved outside of the window that it was in. The object should respond by calling **WinChangeAck()**, which will cause enter and leave events to be generated for all windows affected by the pointer's change.

Source: Window system (**WinMovePtr()**).

Destination: System Input object (usually the GenSystem object).

Interception: Must be handled via call to **WinChangeAck()**.

Parameters: None.

Return: Nothing.

■ MSG_META_IMPLIED_WIN_CHANGE

```
@importMessage MetaWindowMessages, void MSG_META_IMPLIED_WIN_CHANGE(
        optr                inputObj,
        WindowHandle        ptrWin);
```

Sent to the System Input Object (Normally the UI's GenSystem obj) in response to a call to **WinChangeAck()**, to inform it which window the mouse has moved into. The system input object is responsible for passing this message on to the Input object of affected geodes.

Source: Window system (**WinChageAck()**).

Destination: Initially System Input Object (usually the **GenSystemClass** object), though is relayed on to Geode Input Object (usually a **GenApplicationClass** object).

Interception: May be intercepted to learn when an implied window change has occurred, but subclasses should not change any default functionality.

Parameters: *inputObj* Window which has implied grab (or zero if there is no implied grab).

ptrWin Window that pointer is in.

Return: Nothing.

■ MSG_META_RAW_UNIV_ENTER

```
@importMessage MetaWindowMessages, void MSG_META_RAW_UNIV_ENTER(
         optr                 inputObj,
         WindowHandle         ptrWin);
```

This message is generated by the window system whenever the mouse crosses into a window. This message is sent to the window's input object. This is sent whenever the mouse pointer crosses a window boundary, regardless of any existing window grab.

Source: Window system (**WinChangeAck()**).

Destination: Initially System input object (usually the **GenSystemClass** object), though is relayed on to Geode Input Object (usually a **GenApplicationClass** object), and finally onto individual Window's Input Object.

Interception: May be intercepted to track current status of whether mouse is within window or not. Specific UIs rely on these messages to control auto-raise, click-to-raise arming, and correct implied and active mouse grab interaction behavior.

Parameters: *inputObj* Input Object of window method refers to.

ptrWin Window that method refers to.

Return: Nothing.

■ MSG_META_RAW_UNIV_LEAVE

```
@importMessage MetaWindowMessages, void MSG_META_RAW_UNIV_LEAVE(
         optr                 inputObj,
         WindowHandle         ptrWin);
```

This message is generated by the window system whenever the mouse crosses out of a window. This message is sent to the window's input object. This is sent whenever the mouse pointer crosses a window boundary, regardless of any existing window grab.

Source: Window system (**WinChangeAck()**).

Destination: Initially System Input Object (usually the **GenSystemClass** object), though is relayed on to Geode Input Object (usually a **GenApplicationClass** object), and finally onto individual Window's Input Object.

Interception: May be intercepted to track current status of whether mouse is within window or not. Specific UIs rely on these messages to control auto-raise, click-to-raise arming, and correct implied and active mouse grab interaction behavior.

Parameters:

inputObj	Input Object of window method refers to.
ptrWin	Window that method refers to.

Return: Nothing.

1.1.3.2 Input Messages

These messages contain "raw" input events; events which have not yet been processed by the Input Manager. Most applications intercepting input events should intercept events which have been so processed, as described in section 1.1.3.3 on page 88.

■ MSG_META_MOUSE_BUTTON

```
@importMessage MetaInputMessages, void MSG_META_BUTTON(
        word   xPosition,
        word   yPosition,
        word   inputState);
```

This message is sent out on any button press or release.

Parameters:

xPosition	X-coordinate of mouse event.
yPosition	Y-coordinate of mouse event.
inputState	High byte is a ShiftState; low byte is ButtonInfo.

Return: Nothing.

■ MSG_META_MOUSE_PTR

```
@importMessage MetaInputMessages, void MSG_META_PTR(
         word  xPosition,
         word  yPosition,
         word  inputState);
```

This message is sent out on any mouse movement.

Parameters: *xPosition* X-coordinate of mouse event.

yPosition Y-coordinate of mouse event.

inputState High byte is a ShiftState; low byte is ButtonInfo.

Return: Nothing.

■ MSG_META_KBD_CHAR

```
@importMessage MetaInputMessages, void MSG_META_KBD_CHAR(
         word  character,
         word  flags, /* low byte = CharFlags, high byte = ShiftState */
         word  state);/* low byte = ToggleState, high byte = scan code */
```

This is the message sent out on any keyboard press or release. To determine whether the message is in response to a press or a release, check the CF_RELEASE bit of the *flags* field.

Parameters: *character* Low byte contains **Char** value of incoming character.

flags High byte is **ShiftState**; low byte is **CharFlags**.

state High byte is raw PC scan code; low byte is **ToggleState**.

Return: Nothing.

■ MSG_META_MOUSE_DRAG

```
@importMessage MetaInputMessages, void MSG_META_MOUSE_DRAG(
         word  xPosition,
         word  yPosition,
         word  inputState);
```

This is a very low-level message, signalling that the user is dragging the mouse.

Parameters: *xPosition* X-coordinate of mouse event.

yPosition Y-coordinate of mouse event.

inputState High byte is a **ShiftState**; low byte is **ButtonInfo**.

Return: Nothing.

1.1.3.3 UI Messages

The User Interface generates many messages which may alert objects to events which will allow them to work with the user. These events include those generated from the actions of input devices and clipboard-related events.

1.1

Clipboard Messages

The following messages are used to implement common clipboard functions.

■ MSG_META_CLIPBOARD_CUT

```
@importMessage MetaUIMessages, void MSG_META_CLIPBOARD_CUT();
```

This message is sent to an object which is supposed to be the destination of a clipboard operation. MSG_META_CLIPBOARD_CUT should register the current selection with the UI as the new clipboard item, but also delete the current selection.

Source: Sent by anyone to perform clipboard operation.

Destination: Object which will support clipboard operations. By default, a GenEditControl sends this message to the targeted object.

Interception: May be intercepted to add clipboard support to existing class that doesn't currently have clipboard support or to enhance or replace functionality of object that does support the clipboard.

Parameters: None.

Return: Nothing.

■ MSG_META_CLIPBOARD_COPY

```
@importMessage MetaUIMessages, void MSG_META_CLIPBOARD_COPY();
```

This message is sent to an object which is supposed to be the destination of a clipboard operation. MSG_META_CLIPBOARD_COPY should be handled by registering the current selection with UI as the new clipboard item.

Source: Sent by anyone to perform clipboard operation.

Destination: Object which will support clipboard operations. By default, a GenEditControl sends this message to the targeted object.

Interception: May be intercepted to add clipboard support to existing class that doesn't currently have clipboard support or to enhance or replace functionality of object that does support the clipboard.

Parameters: None.

Return: Nothing.

■ MSG_META_CLIPBOARD_PASTE

```
@importMessage MetaUIMessages, void MSG_META_CLIPBOARD_PASTE();
```

1.1

This message is sent to an object which is supposed to be the destination of a clipboard operation. MSG_META_CLIPBOARD_PASTE should replace the current selection with the current clipboard item, which can be obtained from the UI.

Source: Sent by anyone to perform clipboard operation.

Destination: Object which will support clipboard operations. By default, a GenEditControl sends this message to the targeted object.

Interception: May be intercepted to add clipboard support to existing class that doesn't currently have clipboard support or to enhance or replace functionality of object that does support the clipboard.

Parameters: None.

Return: Nothing.

■ MSG_META_CLIPBOARD_NOTIFY_QUICK_TRANSFER_FEEDBACK

```
@importMessage MetaUIMessages, void
               MSG_META_CLIPBOARD_NOTIFY_QUICK_TRANSFER_FEEDBACK(
        ClipboardQuickNotifyFlags flags);
```

This message is sent to the source of a quick transfer item when a potential destination provides feedback to the user indicating whether a move, a copy or no operation will occur. The default behavior is determined by the destination, but the user may be able to override with the MOVE or COPY override keys.

Source: Sent by quick-transfer mechanism.

Destination: Sent to optr passed to **ClipboardStartQuickTransfer()**. Handled if the quick-transfer source needs to know what quick-transfer operation a potential destination will perform. Handler need not call superclass.

Interception: Message sent directly to destination, no need to intercept.

Parameters: *flags* Quick transfer cursor action specified by source (see **ClipboardSetQuickTransferFeedback()**).

Return: Nothing.

■ MSG_META_CLIPBOARD_NOTIFY_QUICK_TRANSFER_CONCLUDED

```
@importMessage MetaUIMessages, void
            MSG_META_CLIPBOARD_NOTIFY_QUICK_TRANSFER_FEEDBACK(
        ClipboardQuickNotifyFlags flags);
```

1.1

This message is sent to the source of a quick transfer item when the operation is completed. The **ClipboardQuickNotifyFlags** are set by any MSG_META_END_MOVE_COPY handler. This is only sent out if the source requests notification with the CQTF_NOTIFICATION flag passed to **ClipboardStartQuickTransfer()**.

Source: Sent by quick-transfer mechanism.

Destination: Sent to optr passed to **ClipboardStartQuickTransfer()**. Handled if the quick-transfer source needs to know what quick-transfer operation was performed. Handler need not call superclass.

Interception: Message sent directly to source of transfer; no need to intercept.

Parameters: *flags* Quick transfer cursor action specified by source (see **ClipboardSetQuickTransferFeedback()**).

Return: Nothing.

■ MSG_META_CLIPBOARD_NOTIFY_TRANSFER_ITEM_FREED

```
@importMessage MetaUIMessages, void
            MSG_META_CLIPBOARD_NOTIFY_TRANSFER_ITEM_FREED(
        VMFileHandle itemFile,
        VMBlockHandle itemBlock);
```

Sent to all ODs in Transfer Notify List to help maintain integrity of transfer items from VM files other than the UI's transfer VM file. Only sent if VM file handle of transfer item that is being freed is different from UI's transfer VM file handle. If a transfer item from a VM file other than the UI's transfer VM file is registered, the VM blocks in that transfer item cannot be freed and the VM file cannot be closed until notification is sent saying that the transfer item has been freed. Registrars of such transfer items should keep track of the VM file handle and VM block handle of the item to check against the info sent by this message.

Source: Sent by the clipboard mechanism.

◆Objects

Destination: Sent to optrs on transfer notification list, added with **ClipboardAddToNotificationList()**. Handled if clipboard changes need to be monitored.

Interception: Unlikely.

Parameters:

itemFile	File containing the transfer item.
itemBlock	Block containing the transfer item.

Return: Nothing.

■ MSG_META_CLIPBOARD_NOTIFY_NORMAL_TRANSFER_ITEM_CHANGED

```
@importMessage MetaUIMessages, void
                    MSG_META_CLIPBOARD_NOTIFY_NORMAL_TRANSFER_ITEM_CHANGED();
```

Sent to all ODs in Transfer Notify List to help with updating of Cut, Copy, and Paste button states. Recipients can call **ClipboardQueryItem()** to check if the new normal transfer item contains formats that the recipient supports. If not, Paste button can be disabled.

Source: Sent by the clipboard mechanism, relayed by GenEditControl.

Destination: Sent to optrs on transfer notification list, added with **ClipboardAddToNotificationList()**. Handled if clipboard changes need to be monitored.

Interception: Unlikely.

Parameters: None.

Return: Nothing.

Undo Messages

These messages implement the "undo" mechanism which allows objects to store a chain of actions which can later be undone. For more information about Undo, see "GenProcessClass" on page 135.

■ MSG_META_UNDO

```
@importMessage MetaUIMessages, void MSG_META_UNDO(
        AddUndoActionStruct *data);
```

This message is sent to an object which is supposed to be the destination of a clipboard operation.

Source: Sent by anyone to perform clipboard operation.

Destination: Object which will support clipboard operations. By default, a GenEditControl sends this message to the targeted object.

Interception: May be intercepted to add clipboard support to existing class that doesn’t currently have clipboard support or to enhance or replace functionality of object that does support the clipboard.

Parameters: None.

Return: Nothing.

1.1

■ MSG_META_UNDO_FREEING_ACTION

```
@importMessage MetaUIMessages, void MSG_META_UNDO_FREEING_ACTION(
        AddUndoActionStruct *data);
```

This message is sent to an object which is supposed to be the destination of a clipboard operation. This message is used to undo those actions which may free an important block of memory.

Source: Sent by anyone to perform clipboard operation.

Destination: Object which will support clipboard operations. By default, a GenEditControl sends this message to the targeted object.

Interception: May be intercepted to add clipboard support to existing class that doesn’t currently have clipboard support or to enhance or replace functionality of object that does support the clipboard.

Parameters: None.

Return: Nothing.

■ MSG_META_SELECT_ALL

```
@importMessage MetaUIMessages, void MSG_META_SELECT_ALL();
```

This message is sent to an object which is supposed to be the destination of a clipboard operation.

Source: Sent by anyone to perform clipboard operation.

Destination: Object which will support clipboard operations. By default, a GenEditControl sends this message to the targeted object.

Interception: May be intercepted to add clipboard support to existing class that doesn’t currently have clipboard support or to enhance or replace functionality of object that does support the clipboard.

Parameters: None.

Objects

Return: Nothing.

■ MSG_META_DELETE

```
@importMessage MetaUIMessages, void MSG_META_DELETE();
```

This message is sent to an object which is supposed to be the destination of a clipboard operation.

Source: Sent by anyone to perform clipboard operation.

Destination: Object which will support clipboard operations. By default, a GenEditControl sends this message to the targeted object.

Interception: May be intercepted to add clipboard support to existing class that doesn't currently have clipboard support or to enhance or replace functionality of object that does support the clipboard.

Parameters: None.

Return: Nothing.

Input Messages

These are perhaps the most often intercepted messages, allowing objects to detect input events.

■ MSG_META_GAINED_MOUSE_EXCL

```
@importMessage MetaUIMessages, void MSG_META_GAINED_MOUSE_EXCL();
```

The object will receive this message when it has received the mouse exclusive.

■ MSG_META_LOST_MOUSE_EXCL

```
@importMessage MetaUIMessages, void MSG_META_LOST_MOUSE_EXCL();
```

The object will receive this message when it has lost the mouse exclusive.

■ MSG_META_GAINED_KBD_EXCL

```
@importMessage MetaUIMessages, void MSG_META_GAINED_KBD_EXCL();
```

The object will receive this message when it has received the keyboard exclusive.

■ MSG_META_LOST_KBD_EXCL

@importMessage MetaUIMessages, void `MSG_META_LOST_KBD_EXCL();`

The object will receive this message when it has lost the keyboard exclusive.

■ MSG_META_GAINED_PRESSURE_EXCL

@importMessage MetaUIMessages, void `MSG_META_GAINED_PRESSURE_EXCL();`

1.1

The object will receive this message when it has received the pressure exclusive, meaning it will get certain low-level mouse events.

■ MSG_META_LOST_PRESSURE_EXCL

@importMessage MetaUIMessages, void `MSG_META_LOST_PRESSURE_EXCL();`

The object will receive this message when it has lost the pressure exclusive.

■ MSG_META_GAINED_DIRECTION_EXCL

@importMessage MetaUIMessages, void `MSG_META_GAINED_DIRECTION_EXCL();`

The object will receive this message when it has received the direction exclusive, meaning it will get certain low-level mouse events.

■ MSG_META_LOST_DIRECTION_EXCL

@importMessage MetaUIMessages, void `MSG_META_LOST_DIRECTION_EXCL();`

The object will receive this message when it has lost the direction exclusive.

Hierarchical Messages

These messages allow object to detect changes in the makeup of the three hierarchies which affect the paths of input and actions within the system: the Focus, Target, and Model hierarchies. These hierarchies are discussed in “Input,” Chapter 11 of the Concepts Book.

■ MSG_META_GRAB_FOCUS_EXCL

@importMessage MetaUIMessages, void `MSG_META_GRAB_FOCUS_EXCL();`

May be passed to any visible or generic object to cause it to become the active focus within its focus level. The leaf object in the hierarchy which gains the focus exclusive will automatically be given the keyboard exclusive, and will thereby receive MSG_META_KBD_CHAR events that follow.

◆ Objects

Commonly sent to text objects and other gadgets to switch the current focus. May also be passed to GenPrimarys, GenDisplays, independently realizable GenInteractions, GenDisplayControl, GenViews, etc. (windowed things) to cause them to become the active focus window within their level of the focus hierarchy, if possible (specific UI's having real-estate focus, for instance, would ignore this request).

Note that the object will not actually gain the focus exclusive until *all* other nodes above it in the hierarchy also have the focus in their levels.

This is the message equivalent of HINT_DEFAULT_FOCUS on generic objects.

■ MSG_META_RELEASE_FOCUS_EXCL

```
@importMessage MetaUIMessages, void MSG_META_RELEASE_FOCUS_EXCL();
```

Opposite of MSG_META_GRAB_FOCUS_EXCL. If the object does not currently have the exclusive, nothing will be done.

■ MSG_META_GET_FOCUS_EXCL

```
@importMessage MetaUIMessages, Boolean MSG_META_GET_FOCUS_EXCL(
        optr   *focusObject);
```

May be sent to any visible or generic object which is a focus node, to get current focus object directly below the node, if any, regardless of whether current node is active (has the exclusive itself).

Focus nodes in Generic UI library: GenSystem, GenField, GenApplication, GenPrimary, GenDisplayGroup, GenDisplay, GenView, GenInteraction (independently displayable only). Focus nodes in Visible UI library: VisContent.

Parameters: *focusObject* This will be filled with return value, the focus object below the object receiving the message.

Return: Will return *true* if message responded to. Will return *false* if the message was sent to an object which is not a focus node.

focusObject The focus node under the object receiving the message.

Warnings: This is a bad way to go about sending a message to currently active objects. For example, if you call from the application thread to the UI thread to find out which is the current focus gadget and then send a message to it, it is possible for the active gadget to change between the two calls. Use MSG_META_SEND_CLASSED_EVENT for this type of operation if at all possible.

■ MSG_META_GRAB_TARGET_EXCL

```
@importMessage MetaUIMessages, void MSG_META_GRAB_TARGET_EXCL();
```

May be passed to any visible or generic object to cause it to become the active target within the target level that it is in. The active target hierarchy is the path for the transmission of messages via TO_TARGET request of MSG_META_SEND_CLASSED_EVENT.

Commonly sent to text objects and views to switch which is the current target. May also be passed to GenPrimarys, GenDisplays, independently realizable GenInteractions, GenDisplayControl, GenViews, etc. (windowed things) to cause them to become the active target window within their level f the target hierarchy.

The specific UI will automatically grab the Target exclusive for an object on any mouse press within the object if it is marked as GA_TARGETABLE.

Note that the object will not actually gain the target exclusive until *all* other nodes above it in the hierarchy also have the target exclusive within their levels. This is the message equivalent of HINT_DEFAULT_TARGET.

■ MSG_META_RELEASE_TARGET_EXCL

```
@importMessage MetaUIMessages, void MSG_META_RELEASE_TARGET_EXCL();
```

Opposite of MSG_META_GRAB_TARGET_EXCL. If the object does not currently have the exclusive, nothing will be done.

■ MSG_META_GET_TARGET_EXCL

```
@importMessage MetaUIMessages, void MSG_META_GET_TARGET_EXCL(
         optr targetObject);
```

May be sent to any visible or generic object which is a target node, to get the current target object directly below the node, if any, regardless of whether the current node is active (has the exclusive itself).

Target nodes in Generic UI library: GenSystem, GenField, GenApplication, GenPrimary, GenDisplay, GenView, GenInteraction (independently displayable only). Target nodes in Visible UI library: VisContent.

◆ Objects

Parameters: *targetObject* This will be filled with return value, the target object below the object receiving the message.

Return: Will return *true* if message responded to. Will return *false* if the message was sent to an object which is not a target node.

targetObject The target node under the object receiving the message.

Warnings: This is a bad way to go about sending a message to currently active objects. For example, if you call from the application thread to the UI thread to find out which is the current target display, and then send a message to it, it is possible for the active display to change between the two calls. Use MSG_META_SEND_CLASSED_EVENT for this type of operation if at all possible.

■ MSG_META_GRAB_MODEL_EXCL

```
@importMessage MetaUIMessages, void MSG_META_GRAB_MODEL_EXCL();
```

May be passed to any visible or generic object to cause it to become the active model within the model level that it is in. The active model hierarchy is the override path for the transmission of messages via TO_MODEL of MSG_META_SEND_CLASSED_EVENT. (If no model hierarchy exists, the messages will be sent down the Target hierarchy.)

Note that the object will not actually gain the model exclusive until *all* other nodes above it in the hierarchy also have the model exclusive within their levels. This is the message equivalent of HINT_MAKE_DEFAULT_MODEL.

■ MSG_META_RELEASE_MODEL_EXCL

```
@importMessage MetaUIMessages, void MSG_META_RELEASE_MODEL_EXCL();
```

Opposite of MSG_META_GRAB_MODEL_EXCL. If the object does not currently have the exclusive, nothing will be done.

■ MSG_META_GET_MODEL_EXCL

```
@importMessage MetaUIMessages, void MSG_META_GET_MODEL_EXCL(
         optr targetObject);
```

May be sent to any visible or generic object which is a model node, to get current model object directly below the node, if any, regardless of whether current node is active (has the exclusive itself).

Model nodes in Generic UI library: GenSystem, GenApplication, GenDocumentControl, GenDocumentGroup.

Parameters: *modelObject* This will be filled with return value, the target object below the object receiving the message.

Return: Will return *true* if message responded to. Will return *false* if the message was sent to an object which is not a target node.

modelObject The target node under the object receiving the message.

Warnings: This is a bad way to go about sending a message to currently active objects. For example, if you call from the application thread to the UI thread to find out which is the current model display, and then send a message to it, it is possible for the active display to change between the two calls. Use MSG_META_SEND_CLASSED_EVENT for this type of operation if at all possible.

■ MSG_META_GAINED_FOCUS_EXCL

@importMessage MetaUIMessages, void MSG_META_GAINED_FOCUS_EXCL();

See description for this and other gained/lost exclusive messages below.

Special note on MSG_META_GAINED_FOCUS_EXCL and MSG_META_LOST_FOCUS_EXCL: If the object receiving MSG_META_GAINED_FOCUS_EXCL is the leaf object in the hierarchy, meaning that it is either not a node itself, or if it is a node, does not have any object below it which has grabbed the exclusive, then the object will automatically be granted the MSG_META_GAINED_KBD_EXCL as well, and thereby receive any MSG_META_KBD_CHAR messages which are generated. The object will receive MSG_META_LOST_KBD_EXCL before MSG_META_LOST_FOCUS_EXCL.

■ MSG_META_LOST_FOCUS_EXCL

@importMessage MetaUIMessages, void MSG_META_LOST_FOCUS_EXCL();

See description for this and other gained/lost exclusive messages below.

Special note on MSG_META_GAINED_FOCUS_EXCL and MSG_META_LOST_FOCUS_EXCL: If the object receiving MSG_META_GAINED_FOCUS_EXCL is the leaf object in the hierarchy, meaning that it is either not a node itself, or if it is a node, does not have any object below it which has grabbed the exclusive, then the object will automatically be granted the MSG_META_GAINED_KBD_EXCL as well, and thereby receive any MSG_META_KBD_CHAR messages which are generated. The object will receive MSG_META_LOST_KBD_EXCL before MSG_META_LOST_FOCUS_EXCL.

■ MSG_META_GAINED_SYS_FOCUS_EXCL

```
@importMessage MetaUIMessages, void MSG_META_GAINED_SYS_FOCUS_EXCL();
```

See description for this and other gained/lost exclusive messages below.

■ MSG_META_LOST_SYS_FOCUS_EXCL

```
@importMessage MetaUIMessages, void MSG_META_LOST_SYS_FOCUS_EXCL();
```

See description for this and other gained/lost exclusive messages below.

■ MSG_META_GAINED_TARGET_EXCL

```
@importMessage MetaUIMessages, void MSG_META_GAINED_TARGET_EXCL();
```

See description for this and other gained/lost exclusive messages below.

■ MSG_META_LOST_TARGET_EXCL

```
@importMessage MetaUIMessages, void MSG_META_LOST_TARGET_EXCL();
```

See description for this and other gained/lost exclusive messages below.

■ MSG_META_GAINED_SYS_TARGET_EXCL

```
@importMessage MetaUIMessages, void MSG_META_GAINED_SYS_TARGET_EXCL();
```

See description for this and other gained/lost exclusive messages below.

■ MSG_META_LOST_SYS_TARGET_EXCL

```
@importMessage MetaUIMessages, void MSG_META_LOST_SYS_TARGET_EXCL();
```

See description for this and other gained/lost exclusive messages below.

■ MSG_META_GAINED_MODEL_EXCL

```
@importMessage MetaUIMessages, void MSG_META_GAINED_MODEL_EXCL();
```

See description for this and other gained/lost exclusive messages below.

■ MSG_META_LOST_MODEL_EXCL

```
@importMessage MetaUIMessages, void MSG_META_LOST_MODEL_EXCL();
```

See description for this and other gained/lost exclusive messages below.

■ MSG_META_GAINED_SYS_MODEL_EXCL

```
@importMessage MetaUIMessages, void MSG_META_GAINED_SYS_MODEL_EXCL();
```

See description for this and other gained/lost exclusive messages below.

■ MSG_META_LOST_SYS_MODEL_EXCL

```
@importMessage MetaUIMessages, void MSG_META_LOST_SYS_MODEL_EXCL();
```

1.1

These paired gained/lost messages for the Focus, Target, and Model hierarchies are sent, always in the order GAINED, then at some point LOST, to objects on the hierarchy. The GAINED message is sent only when the object in question and all nodes in the hierarchy above that object have gained the exclusive from the node above them, all the way up to the application object.

In other words, just grabbing the exclusive from the next node up doesn't always guarantee you'll get a GAINED message; the node you're grabbing from must have itself received a GAINED message but not yet the LOST message. Your object will receive the LOST message if it has either released the exclusive, or the node from which you grabbed the exclusive itself received a LOST message.

The GAINED_SYS and LOST_SYS messages behave similarly, except that an object can only gain the SYS_EXCL (System exclusive) if it and all nodes above it to the GenSystem object have the grab from the next node up. An object will never receive a GAINED_SYS_EXCL message if it has not already received an (Application) GAINED_EXCL message also. Similarly, an object will always receive a LOST_SYS_EXCL message before it receives an (Application) LOST_EXCL message.

Source: Do *not* send these messages to objects yourself, unless you are implementing or extending the above mechanism. These messages should be sent only by the node object which is above the object receiving the message.

Destination: Any **MetaClass** object which has grabbed and not yet released the focus exclusive.

Interception: Generic UI objects, **VisTextClass**, and all node objects provide default behavior for processing this message. If you intercept above any of these levels, be sure to call the superclass to let these objects know the exclusive has been gained.

◆Objects

Miscellaneous Input Messages

■ MSG_META_GRAB_KBD

```
@importMessage MetaUIMessages, void MSG_META_GRAB_KBD();
```

This message grabs the keyboard for an object. The grab will not be taken away from another object if it currently has the keyboard grab. To forcefully grab the keyboard in this case, use MSG_META_FORCE_GRAB_KBD.

■ MSG_META_FORCE_GRAB_KBD

```
@importMessage MetaUIMessages, void MSG_META_FORCE_GRAB_KBD();
```

This message forcefully grabs the keyboard for an object, tasking the grab away from another object, if necessary.

■ MSG_META_RELEASE_KBD

```
@importMessage MetaUIMessages, void MSG_META_RELEASE_KBD();
```

This message releases the keyboard grab for an object.

■ MSG_META_RELEASE_FT_EXCL

```
@importMessage MetaUIMessages, void MSG_META_RELEASE_FT_EXCL();
```

This message releases exclusive(s) that the object may have on the Focus and Target hierarchies.

■ MSG_META_GAINED_DEFAULT_EXCL

```
@importMessage MetaUIMessages, void MSG_META_GAINED_DEFAULT_EXCL();
```

Sent out in response to this object receiving MSG_VIS_VUP_QUERY with SVQT_TAKE_DEFAULT_EXCLUSIVE, to notify a GenTrigger that it has gained the default exclusive.

■ MSG_META_LOST_DEFAULT_EXCL

```
@importMessage MetaUIMessages, void MSG_META_LOST_DEFAULT_EXCL();
```

Sent out in response to this object receiving MSG_VIS_VUP_QUERY with SVQT_RELEASE_DEFAULT_EXCLUSIVE, to notify a GenTrigger that it has lost the default exclusive.

■ MSG_META_GAINED_FULL_SCREEN_EXCL

```
@importMessage MetaUIMessages, void MSG_META_GAINED_FULL_SCREEN_EXCL();
```

This message is sent to GenFields or GenApplications upon gain of the "full-screen" exclusive. The full-screen exclusive grants the object the top screen-dominating object at its level.

■ MSG_META_LOST_FULL_SCREEN_EXCL

```
@importMessage MetaUIMessages, void MSG_META_LOST_FULL_SCREEN_EXCL();
```

This message is sent to GenFields or GenApplications upon loss of the "full-screen" exclusive. The full-screen exclusive grants the object the top screen-dominating object at its level.

■ MSG_META_MOUSE_BUMP_NOTIFICATION

```
@importMessage MetaUIMessages, void MSG_META_MOUSE_BUMP_NOTIFICATION(
        sword  xBump,
        sword  yBump);
```

This message is an event that the input manager places in the input queue to notify the UI that it has bumped the mouse position past this point in the queue. This method is sent only when **IMBumpMouse()** is called.

Parameters:		
	xBump	Horizontal relative bump.
	yBump	Vertical relative bump.

■ MSG_META_FUP_KBD_CHAR

```
@importMessage MetaUIMessages, Boolean MSG_META_FUP_KBD_CHAR(
        word   character,
        word   flags,
        word   state);
```

When a leaf object in the focus hierarchy gets a MSG_META_KBD_CHAR, and does not care about the character, it sends this message to itself to see if a parent object wants to handle it.

Parameters:		
	character	The low byte contains a **Char** value.
	flags	High byte is a **ShiftState** field; low byte is a **CharFlags** field.
	inputState	High byte is the raw PC scan code; low byte is a **ToggleState** field.

◆Objects

Return: Will return *true* if the character was handled by someone (and should not be used elsewhere).

■ MSG_META_PRE_PASSIVE_KBD_CHAR

```
@importMessage MetaUIMessages, KbdReturnFlags
                    MSG_META_PRE_PASSIVE_KBD_CHAR(
        word        character,
        word        flags,
        word        state);
```

This message sends a keyboard character to any object requesting preview of the keyboard events.

Parameters: *character* The low byte contains a **Char** value.

flags High byte is a **ShiftState** field; low byte is a **CharFlags** field.

inputState High byte is the raw PC scan code; low byte is a **ToggleState** field.

Return: Flags field specifying what should happen to event.

Structures:

```
typedef WordFlags KbdReturnFlags;

#define KRF_PREVENT_PASS_THROUGH 0x8000
/* Set for passive keyboard routines if event should
 * be destroyed and not passed on to implied or
 * default grab. */
```

■ MSG_META_POST_PASSIVE_KBD_CHAR

```
@importMessage MetaUIMessages, KbdReturnFlags
                    MSG_META_POST_PASSIVE_KBD_CHAR(
        word        character,
        word        flags,
        word        state);
```

This message passes keyboard characters to all objects having registered interest in getting keyboard events after they have been handled.

Parameters: *character* The low byte contains a **Char** value.

flags High byte is a **ShiftState** field; low byte is a **CharFlags** field.

inputState High byte is the raw PC scan code; low byte is a **ToggleState** field.

Return: Flags field specifying what should happen to event.

Structures:

```
typedef WordFlags KbdReturnFlags;
#define KRF_PREVENT_PASS_THROUGH 0x8000
/* Set for passive keyboard routines if event should
 * be destroyed and not passed on to implied or
 * default grab. */
```

1.1

■ MSG_META_QUERY_IF_PRESS_IS_INK

```
@importMessage MetaUIMessages, InkReturnValue
                MSG_META_QUERY_IF_PRESS_IS_INK(
          sword  xPosition,
          sword  yPosition);
```

Return whether or not a MSG_META_START_SELECT should be passed on to the object, or whether it should be intercepted and turned into ink.

Source: Sent by any object (usually VisComp) to determine if one of its children wants ink.

Destination: Any object in the Vis linkage that may be clicked on with the mouse.

Interception: The default handler returns IRV_NO_INK. Objects that want presses to be turned into ink need to return IRV_DESIRES_INK. Some objects that need to do work on another thread (such as a GenView) to determine whether the press should be ink or not can return IRV_WAIT, which holds up the MSG_META_START_SELECT until a MSG_GEN_APPLICATION_INK_QUERY_REPLY is sent to the application object. By default, clicks on VisComp-derived objects will *not* be ink. To change this, set **VisCompMakePressesInk()** as the handler for this message.

Parameters: *xPosition* X-coordinate of selection start.

yPosition Y-coordinate of selection start.

Return: Indication whether object thinks the press was ink.

■ MSG_META_LARGE_QUERY_IF_PRESS_IS_INK

```
@importMessage MetaUIMessages, void
                MSG_META_LARGE_QUERY_IF_PRESS_IS_INK(
          InkReturnParams     *retVal,
          LargeMouseData      *largeMouseDataStruct);
```

This message is sent by the system to children with the VCNA_LARGE_DOCUMENT_MODEL bit set to determines whether or not a MSG_META_LARGE_START_SELECT should be processed as ink.

Source: Sent by any object (usually VisComp) to determine if one of its children wants ink.

Destination: Any object in the Vis linkage that may be clicked on with the mouse.

Parameters: *retVal* Pointer to an **InkReturnParams** structure that will be filled in by the handler for this message.

largeMouseDataStruct Pointer to a **LargeMouseData** struct that stores information about the large mouse event.

Interception: The default handler returns IRV_NO_INK. Objects that want presses to be turned into ink need to return IRV_DESIRES_INK. Some objects that need to do work on another thread (such as a VisContent) to determine whether the press should be ink or not can return IRV_WAIT, which holds up the MSG_META_LARGE_START_SELECT until a MSG_GEN_APPLICATION_INK_QUERY_REPLY is sent to the application object.

Mouse Input Messages

The following messages allow an application to detect the nature and behavior of pointing devices within the system.

■ MSG_META_START_SELECT

```
@importMessage MetaUIMessages, void MSG_META_START_SELECT(
          MouseReturnParams   *retVal,
          sword               xPosition,
          sword               yPosition,
          word,               inputState);
```

For description of this and other button messages, see below.

■ MSG_META_END_SELECT

```
@importMessage MetaUIMessages, void MSG_META_END_SELECT(
          MouseReturnParams   *retVal,
          sword               xPosition,
          sword               yPosition,
          word,               inputState);
```

For description of this and other button messages, see below.

1.1

■ MSG_META_START_MOVE_COPY

```
@importMessage MetaUIMessages, void MSG_META_START_MOVE_COPY(
          MouseReturnParams   *retVal,
          sword               xPosition,
          sword               yPosition,
          word,               inputState);
```

For description of this and other button messages, see below.

■ MSG_META_END_MOVE_COPY

```
@importMessage MetaUIMessages, void MSG_META_END_MOVE_COPY(
          MouseReturnParams   *retVal,
          sword               xPosition,
          sword               yPosition,
          word,               inputState);
```

For description of this and other button messages, see below.

■ MSG_META_START_FEATURES

```
@importMessage MetaUIMessages, void MSG_META_START_FEATURES(
          MouseReturnParams   *retVal,
          sword               xPosition,
          sword               yPosition,
          word,               inputState);
```

For description of this and other button messages, see below.

■ MSG_META_END_FEATURES

```
@importMessage MetaUIMessages, void MSG_META_END_FEATURES(
          MouseReturnParams   *retVal,
          sword               xPosition,
          sword               yPosition,
          word,               inputState);
```

For description of this and other button messages, see below.

◆Objects

■ MSG_META_START_OTHER

```
@importMessage MetaUIMessages, void MSG_META_START_OTHER(
          MouseReturnParams   *retVal,
          sword               xPosition,
          sword               yPosition,
          word,               inputState);
```

For description of this and other button messages, see below.

■ MSG_META_END_OTHER

```
@importMessage MetaUIMessages, void MSG_META_END_OTHER(
          MouseReturnParams   *retVal,
          sword               xPosition,
          sword               yPosition,
          word,               inputState);
```

For description of this and other button messages, see below.

■ MSG_META_DRAG_SELECT

```
@importMessage MetaUIMessages, void MSG_META_DRAG_SELECT(
          MouseReturnParams   *retVal,
          sword               xPosition,
          sword               yPosition,
          word,               inputState);
```

For description of this and other button messages, see below.

■ MSG_META_DRAG_MOVE_COPY

```
@importMessage MetaUIMessages, void MSG_META_DRAG_MOVE_COPY(
          MouseReturnParams   *retVal,
          sword               xPosition,
          sword               yPosition,
          word,               inputState);
```

For description of this and other button messages, see below.

■ MSG_META_DRAG_FEATURES

```
@importMessage MetaUIMessages, void MSG_META_DRAG_FEATURES(
          MouseReturnParams   *retVal,
          sword               xPosition,
          sword               yPosition,
          word,               inputState);
```

For description of this and other button messages, see below.

■ MSG_META_DRAG_OTHER

```
@importMessage MetaUIMessages, void MSG_META_DRAG_OTHER(
          MouseReturnParams   *retVal,
          sword               xPosition,
          sword               yPosition,
          word,               inputState);
```

For description of this and other button messages, see below.

1.1 ■ MSG_META_PRE_PASSIVE_BUTTON

```
@importMessage MetaUIMessages, void MSG_META_PRE_PASSIVE_BUTTON(
          MouseReturnParams   *retVal,
          sword               xPosition,
          sword               yPosition,
          word,               inputState);
```

For description of this and other button messages, see below.

■ MSG_META_POST_PASSIVE_BUTTON

```
@importMessage MetaUIMessages, void MSG_META_POST_PASSIVE_BUTTON(
          MouseReturnParams   *retVal,
          sword               xPosition,
          sword               yPosition,
          word,               inputState);
```

For description of this and other button messages, see below.

■ MSG_META_PRE_PASSIVE_START_SELECT

```
@importMessage MetaUIMessages, void MSG_META_PRE_PASSIVE_START_SELECT(
          MouseReturnParams   *retVal,
          sword               xPosition,
          sword               yPosition,
          word,               inputState);
```

For description of this and other button messages, see below.

■ MSG_META_POST_PASSIVE_START_SELECT

```
@importMessage MetaUIMessages, void MSG_META_POST_PASSIVE_START_SELECT(
          MouseReturnParams   *retVal,
          sword               xPosition,
          sword               yPosition,
          word,               inputState);
```

For description of this and other button messages, see below.

◆Objects

■ MSG_META_PRE_PASSIVE_END_SELECT

```
@importMessage MetaUIMessages, void MSG_META_PRE_PASSIVE_END_SELECT(
         MouseReturnParams   *retVal,
         sword               xPosition,
         sword               yPosition,
         word,               inputState);
```

For description of this and other button messages, see below.

■ MSG_META_POST_PASSIVE_END_SELECT

1.1

```
@importMessage MetaUIMessages, void MSG_META_POST_PASSIVE_END_SELECT(
         MouseReturnParams   *retVal,
         sword               xPosition,
         sword               yPosition,
         word,               inputState);
```

For description of this and other button messages, see below.

■ MSG_META_PRE_PASSIVE_START_MOVE_COPY

```
@importMessage MetaUIMessages, void MSG_META_PRE_PASSIVE_START_MOVE_COPY(
         MouseReturnParams   *retVal,
         sword               xPosition,
         sword               yPosition,
         word,               inputState);
```

For description of this and other button messages, see below.

■ MSG_META_POST_PASSIVE_START_MOVE_COPY

```
@importMessage MetaUIMessages, void
               MSG_META_POST_PASSIVE_START_MOVE_COPY(
         MouseReturnParams   *retVal,
         sword               xPosition,
         sword               yPosition,
         word,               inputState);
```

For description of this and other button messages, see below.

■ MSG_META_PRE_PASSIVE_END_MOVE_COPY

```
@importMessage MetaUIMessages, void MSG_META_PRE_PASSIVE_END_MOVE_COPY(
         MouseReturnParams   *retVal,
         sword               xPosition,
         sword               yPosition,
         word,               inputState);
```

For description of this and other button messages, see below.

■ MSG_META_POST_PASSIVE_END_MOVE_COPY

```
@importMessage MetaUIMessages, void MSG_META_POST_PASSIVE_END_MOVE_COPY(
         MouseReturnParams   *retVal,
         sword               xPosition,
         sword               yPosition,
         word,               inputState);
```

For description of this and other button messages, see below.

1.1

■ MSG_META_PRE_PASSIVE_START_FEATURES

```
@importMessage MetaUIMessages, void MSG_META_PRE_PASSIVE_START_FEATURES(
         MouseReturnParams   *retVal,
         sword               xPosition,
         sword               yPosition,
         word,               inputState);
```

For description of this and other button messages, see below.

■ MSG_META_POST_PASSIVE_START_FEATURES

```
@importMessage MetaUIMessages, void MSG_META_POST_PASSIVE_START_FEATURES(
         MouseReturnParams   *retVal,
         sword               xPosition,
         sword               yPosition,
         word,               inputState);
```

For description of this and other button messages, see below.

■ MSG_META_PRE_PASSIVE_END_FEATURES

```
@importMessage MetaUIMessages, void MSG_META_PRE_PASSIVE_END_FEATURES(
         MouseReturnParams   *retVal,
         sword               xPosition,
         sword               yPosition,
         word,               inputState);
```

For description of this and other button messages, see below.

■ MSG_META_POST_PASSIVE_END_FEATURES

```
@importMessage MetaUIMessages, void MSG_META_POST_PASSIVE_END_FEATURES(
         MouseReturnParams   *retVal,
         sword               xPosition,
         sword               yPosition,
         word,               inputState);
```

For description of this and other button messages, see below.

◆Objects

■ MSG_META_PRE_PASSIVE_START_OTHER

```
@importMessage MetaUIMessages, void MSG_META_PRE_PASSIVE_START_OTHER(
         MouseReturnParams   *retVal,
         sword               xPosition,
         sword               yPosition,
         word,               inputState);
```

For description of this and other button messages, see below.

■ MSG_META_POST_PASSIVE_START_OTHER

```
@importMessage MetaUIMessages, void MSG_META_POST_PASSIVE_START_OTHER(
         MouseReturnParams   *retVal,
         sword               xPosition,
         sword               yPosition,
         word,               inputState);
```

For description of this and other button messages, see below.

■ MSG_META_PRE_PASSIVE_END_OTHER

```
@importMessage MetaUIMessages, void MSG_META_PRE_PASSIVE_END_OTHER(
         MouseReturnParams   *retVal,
         sword               xPosition,
         sword               yPosition,
         word,               inputState);
```

For description of this and other button messages, see below.

■ MSG_META_POST_PASSIVE_END_OTHER

```
@importMessage MetaUIMessages, void MSG_META_POST_PASSIVE_END_OTHER(
         MouseReturnParams   *retVal,
         sword               xPosition,
         sword               yPosition,
         word,               inputState);
```

The above messages are the standard button functions generated by the UI upon receiving MSG_META_BUTTON events from the Input Manager. These messages are sent out to whatever object has the implied grab (whichever window the mouse is over), until the mouse is "grabbed" by an object, after which the messages go there until the mouse is released (ungrabbed).

Parameters:		
	retVal	Structure to hold return values.
	xPosition	X-coordinate of press.
	yPosition	Y-coordinate of press.

inputState High byte is **UIFunctionsActive** structure; low byte is **ButtonInfo** structure.

Structures:

```
typedef struct {
      word              unused;
      MouseReturnFlags  flags;
      optr              ptrImage;
} MouseReturnParameters;
```

```
typedef WordFlags MouseReturnFlags;
/* These flags may be combined using | and &:
      MRF_PROCESSED,
      MRF_REPLAY,
      MRF_PREVENT_PASS_THROUGH,
      MRF_SET_POINTER_IMAGE,
      MRF_CLEAR_POINTER_IMAGE */
```

■ MSG_META_LARGE_PTR

```
@importMessage MetaUIMessages, void MSG_META_LARGE_PTR(
         MouseReturnParams   *retVal,
         LargeMouseData      *largeMouseDataStruct);
```

See below for information about this and other large mouse messages.

■ MSG_META_LARGE_START_SELECT

```
@importMessage MetaUIMessages, void MSG_META_LARGE_START_SELECT(
         MouseReturnParams   *retVal,
         LargeMouseData      *largeMouseDataStruct);
```

See below for information about this and other large mouse messages.

■ MSG_META_LARGE_END_SELECT

```
@importMessage MetaUIMessages, void MSG_META_LARGE_END_SELECT(
         MouseReturnParams   *retVal,
         LargeMouseData      *largeMouseDataStruct);
```

See below for information about this and other large mouse messages.

■ MSG_META_LARGE_START_MOVE_COPY

```
@importMessage MetaUIMessages, void MSG_META_LARGE_START_MOVE_COPY(
         MouseReturnParams   *retVal,
         LargeMouseData      *largeMouseDataStruct);
```

See below for information about this and other large mouse messages.

■ MSG_META_LARGE_END_MOVE_COPY

```
@importMessage MetaUIMessages, void MSG_META_LARGE_END_MOVE_COPY(
         MouseReturnParams   *retVal,
         LargeMouseData      *largeMouseDataStruct);
```

See below for information about this and other large mouse messages.

■ MSG_META_LARGE_START_FEATURES

```
@importMessage MetaUIMessages, void MSG_META_LARGE_START_FEATURES(
         MouseReturnParams   *retVal,
         LargeMouseData      *largeMouseDataStruct);
```

See below for information about this and other large mouse messages.

■ MSG_META_LARGE_END_FEATURES

```
@importMessage MetaUIMessages, void MSG_META_LARGE_END_FEATURES(
         MouseReturnParams   *retVal,
         LargeMouseData      *largeMouseDataStruct);
```

See below for information about this and other large mouse messages.

■ MSG_META_LARGE_START_OTHER

```
@importMessage MetaUIMessages, void MSG_META_LARGE_START_OTHER(
         MouseReturnParams   *retVal,
         LargeMouseData      *largeMouseDataStruct);
```

See below for information about this and other large mouse messages.

■ MSG_META_LARGE_END_OTHER

```
@importMessage MetaUIMessages, void MSG_META_LARGE_END_OTHER(
         MouseReturnParams   *retVal,
         LargeMouseData      *largeMouseDataStruct);
```

See below for information about this and other large mouse messages.

■ MSG_META_LARGE_DRAG_SELECT

```
@importMessage MetaUIMessages, void MSG_META_LARGE_DRAG_SELECT(
         MouseReturnParams   *retVal,
         LargeMouseData      *largeMouseDataStruct);
```

See below for information about this and other large mouse messages.

■ MSG_META_LARGE_DRAG_MOVE_COPY

1.1

```
@importMessage MetaUIMessages, void MSG_META_LARGE_DRAG_MOVE_COPY(
         MouseReturnParams   *retVal,
         LargeMouseData      *largeMouseDataStruct);
```

See below for information about this and other large mouse messages.

■ MSG_META_LARGE_DRAG_FEATURES

```
@importMessage MetaUIMessages, void MSG_META_LARGE_DRAG_FEATURES(
         MouseReturnParams   *retVal,
         LargeMouseData      *largeMouseDataStruct);
```

See below for information about this and other large mouse messages.

■ MSG_META_LARGE_DRAG_OTHER

```
@importMessage MetaUIMessages, void MSG_META_LARGE_DRAG_OTHER(
         MouseReturnParams   *retVal,
         LargeMouseData      *largeMouseDataStruct);
```

Objects which have been set up with 32-bit coordinate spaces must be prepared to handle large mouse events along with regular mouse events.

These messages are available by request for use within 32-bit visible document models. Mouse position data is in full 32-bit integer, 16-bit fraction format, as generated by GenView.

Parameters: *retVal* Structure to hold return values.

largeMouseDataStruct
Structure to hold pass values.

Return: Nothing returned explicitly.

retVal Filled with return values.

Structures:

◆Objects

```
typedef struct {
      PointDWFixed            LMD_location;
      byte                    LMD_buttonInfo;
      UIFunctionsActive       LMD_uiFunctionsActive;
} LargeMouseData;
typedef struct {
      word              unused;
      MouseReturnFlags  flags;
      optr              ptrImage;
      /* Pointer image to use, if MRF_SET_PTR_IMAGE
       * returned */
} MouseReturnParams;
```

■ MSG_META_ENSURE_MOUSE_NOT_ACTIVELY_TRESPASSING

```
@importMessage MetaUIMessages, MouseReturnFlags
               MSG_META_ENSURE_MOUSE_NOT_ACTIVELY_TRESPASSING();
```

Sent to the passive, active, or implied mouse grab chain whenever modality status changes within the system—any object receiving this message which has a window grabbed should make sure that it has a legitimate right to have the window grab active—if not, it should be released (along with the mouse). In particular, menus in stay-up mode should come down, any interaction between the mouse and primary, display, menu, or view windows should be terminated. MSG_GEN_APPLICATION_TEST_WIN_INTERACTABILITY is useful; this message will test any passed OD against the list of window(s) which the mouse is allowed to interact with (Generally, top most system modal window, else top most application modal window, else all windows), and return a flag indicating the result.

Parameters: None.

Return: Flags field which system normally ignores.

■ MSG_META_ENSURE_NO_MENUS_IN_STAY_UP_MODE

```
@importMessage MetaUIMessages, MouseReturnFlags
               MSG_META_ENSURE_NO_MENUS_IN_STAY_UP_MODE();
```

Sent to the passive, active/implied mouse grab chain whenever we want to make sure all of an application's menus are closed. Sent directly to the Flow object from the global shortcut code. Any menus receiving this message which are in stay-up mode should dismiss themselves.

Parameters: None.

Return: Flags field, normally ignored by system.

■ MSG_META_ENSURE_ACTIVE_FT

```
@importMessage MetaUIMessages, void MSG_META_ENSURE_ACTIVE_FT();
```

1.1

Makes sure that some object with the Focus/Target node to which this message may be sent has the Focus and Target exclusives. Called from within the UI, usually when windowed objects below the node have closed, or moved to the back, to give the Focus and/or Target to the most suitable window.

Behavior as implemented in **GenApplicationClass**:
Checks to make sure that something within the application has the Focus and Target exclusives. Called from within the UI, usually on the closure of a window, to give the Focus and/or Target to the next best location.

Typical click-to-type model is implemented using the following rules:
For Target, the priority order is:

1 Anything already having the exclusive.

2 Top targetable PRIO_STD priority level window.

3 Top targetable PRIO_COMMAND priority level window.

For Focus, priority goes to:

1 Anything already having the exclusive.

2 Top system modal window.

3 Top application modal window.

4 Last non-modal window to have or request the exclusive.

5 Window having Target exclusive.

6 Top focusable PRIO_COMMAND priority level window.

Source: Most always internally from the UI, though is unrestricted.

Destination: Focus/Target node, specifically: GenSystem, GenField, or GenApplication object.

◆ Objects

Interception: No reason to intercept. Default behavior is provided by above objects. Could possibly be replaced, but as default behavior varies by specific UI, results could be unpredictable.

Parameters: None.

Return: Nothing.

■ MSG_META_NOTIFY_NO_FOCUS_WITHIN_NODE

```
@importMessage MetaUIMessages, void
              MSG_META_NOTIFY_NO_FOCUS_WITHIN_NODE();
```

Notification from Focus node MSG_META_ENSURE_ACTIVE_FT handler that it was unable to keep/find an object below it suitable for being the focus. The most likely cause is that the last focusable geode/object running below this point has been shut down/closed.

Source: Focus node, MSG_META_ENSURE_ACTIVE_FT handler

Destination: Self

Interception: Intercepted to find something safe to do for the user, such as push this field/application to the back, or mark this object as no longer "focusable" and call MSG_META_ENSURE_ACTIVE_FT on the node above this one, in an attempt to find something for the user to access. If there's nothing left at all in the system, the last focusable application has exited, so it's time to shut down.

Parameters: None.

Return: Nothing.

Miscellaneous Meta Messages

■ MSG_META_UI_FORCE_CONTROLLER_UPDATE

```
@importMessage MetaUIMessages, void MSG_META_UI_FORCE_CONTROLLER_UPDATE(
        ManufacturerID      manufID,
        word                changeID);
```

This message forces an object to update one or all of the GCN notification lists that it communicates with.

Source: Usually sent by a controller to its output.

Destination: Any object.

Parameters: *manufID* **ManufacturerID** of GCN lists.

changeID Notification list ID.
This value may be 0xffffh if all notification lists should be updated or 0xfffeh to generate the standard notifications.

Interception: Objects that send notification for controllers should respond to this message.

■ MSG_META_GEN_PATH_RESTORE_DISK_PROMPT

1.1

```
@importMessage MetaUIMessages, Boolean
            MSG_META_GEN_PATH_RESTORE_DISK_PROMPT(
DiskRestoreError          *error,
GenPathDiskRestoreArgs    *args);
```

This message prompts the user to insert a particular disk into a particular drive when restoring a disk handle for the object's path.

Source: Sent by the callback passed to **DiskRestore()** when a disk handle saved in an object's path is being restored after a shutdown.

Destination: Any object possessing a path.

Parameters: *error* Pointer to store an error code. This error code will be returned to **DiskRestore()**.

args **GenPathDiskRestoreArgs**.

Structures:

```
typedef struct {
      word              GPDRA_pathType;
      word              GPDRA_savedDiskType;
      char              *GPDRA_driveName;
      char              *GPDRA_diskName;
      DiskRestoreError  GPDRA_errorCode;
} GenPathDiskRestoreArgs;
```

GPDRA_pathType stores the vardata tag holding the path.

GPDRA_savedDiskType stores the vardata tag holding the saved disk handle.

GPDRA_driveName and *GPDRA_diskName* store pointers to the null-terminated drive and disk names.

GPDRA_errorCode stores the error code that is returned to **DiskRestore()**.

◆Objects

Interception: May be intercepted if the object has more information to provide to the user, or if the object doesn't wish to prompt the user. If this message is intercepted, it should not call its superclass.

■ MSG_META_PAGED_OBJECT_GOTO_PAGE

```
@importMessage MetaUIMessages, void MSG_META_PAGED_OBJECT_GOTO_PAGE(
        word   page);
```

This message instructs a GenDocument to go to the passed page.

This message is sent out by the GenPageControl object and is handled by a GenApplication's subclassed GenDocument object.

Source: GenPageControl object.

Destination: GenDocument object.

Parameters: *page* Page to set the GenDocument to display.

Interception: You may intercept to provide custom paging behavior.

■ MSG_META_PAGED_OBJECT_NEXT_PAGE

```
@importMessage MetaUIMessages, void MSG_META_PAGED_OBJECT_NEXT_PAGE();
```

This message instructs a GenDocument to go to the next page.

This message is sent out by the GenPageControl object and is handled by a GenApplication's subclassed GenDocument object.

Source: GenPageControl object.

Destination: GenDocument object.

Interception: You may intercept to provide custom paging behavior.

■ MSG_META_PAGED_OBJECT_PREVIOUS_PAGE

```
@importMessage MetaUIMessages, void
              MSG_META_PAGED_OBJECT_PREVIOUS_PAGE();
```

This message instructs a GenDocument to go to the previous page.

This message is sent out by the GenPageControl object and is handled by a GenApplication's subclassed GenDocument object.

Source: GenPageControl object.

Destination: GenDocument object.

Interception: You may intercept to provide custom paging behavior.

■ MSG_META_DELETE_RANGE_OF_CHARS

```
@importMessage MetaUIMessages, void
                 MSG_META_DELETE_RANGE_OF_CHARS(@stack
          VisTextRange        rangeToDelete);
```

1.1

This message instructs an object to delete a range of characters passed in a **VisTextRange**. Generally, this message is sent out when the user crosses out characters within a HWR grid.

Source: GenPenInputControl.

Destination: Any focused object.

Parameters: *rangeToDelete* **VisTextRange** of characters to delete. Objects that are not text objects will need to know how to interpret this value.

Interception: May intercept to provide custom deletion behavior.

■ MSG_META_NOTIFY_TASK_SELECTED

```
@importMessage MetaUIMessages, void MSG_META_NOTIFY_TASK_SELECTED();
```

This message is sent when a task list item of an application in the Express Menu is selected. The default behavior brings the application to the front and gives it the focus.

■ MSG_META_FIELD_NOTIFY_DETACH

```
@importMessage MetaUIMessages, void MSG_META_FIELD_NOTIFY_DETACH(
          optr                field,
          word                shutdownFlag);
```

This message is sent by the GenField object when it is detaching.

Source: GenField.

Destination: The notification destination of the GenField object.

Parameters: *field* Optr of the GenField sending notification.

shutdownFlag *true* if the GenField is detaching because of a shutdown.

Interception: The object receiving notification may handle as desired. As this is a notification only, you should not call the superclass.

◆ Objects

■ MSG_META_FIELD_NOTIFY_NO_FOCUS

```
@importMessage MetaUIMessages, void MSG_META_FIELD_NOTIFY_NO_FOCUS(
                 optr                    field,
                 word                    shutdownFlag);
```

This message is sent by the GenField when it no longer has any applications in the focus hierarchy.

Source: GenField.

Destination: The notification destination of the GenField object.

Parameters: *field* Optr of the GenField sending notification.

shutdownFlag *true* if the GenField lost its focus applications because of a shutdown.

Interception: The object receiving notification may handle as desired. As this is a notification only, you should not call the superclass.

■ MSG_META_FIELD_NOTIFY_START_LAUNCHER_ERROR

```
@importMessage MetaUIMessages, void
                     MSG_META_FIELD_NOTIFY_START_LAUNCHER_ERROR(
                 optr                    field);
```

This message is sent by the GenField when an error occurs while attempting to run the launcher for the field object.

Source: GenField.

Destination: The notification destination of the GenField object.

Parameters: *field* Optr of the GenField sending notification.

Interception: The object receiving notification may handle as desired. As this is a notification only, you should not call the superclass.

■ MSG_META_TEST_WIN_INTERACTIBILITY

```
@importMessage MetaUIMessages, Boolean
                     MSG_META_TEST_WIN_INTERACTIBILITY(
                 optr                    inputOD,
                 WindowHandle            window);
```

This message checks whether a pointing device (usually a mouse) can interact with the passed window.

Source:

Destination: A windowed object.

Parameters: *inputOD* Input optr of the windowed object to check.

window Window to check.

Return: *true* if window is interactable.

■ MSG_META_CHECK_IF_INTERACTIBLE_OBJECT

```
@importMessage MetaUIMessages, Boolean
              MSG_META_CHECK_IF_INTERACTIBLE_OBJECT(
        optr  obj);
```

1.1

This message is sent o objects on the GAGCNLT_ALWAYS_INTERACTABLE_WINDOWS GCN list.

Source: GenApplication object.

Destination: Objects on the GAGCNLT_ALWAYS_INTERACTABLE_WINDOWS GCN list.

Parameters: *obj* Object whose interactable state is being checked.

Return: *true* if object is interactable.

Interception: May intercept.

1.1.3.4 Standard GCN Messages

There are several standard messages which objects adding themselves to the appropriate GCN lists may receive and handle.

■ MSG_NOTIFY_FILE_CHANGE

```
@importMessage MetaGCNMessages, void MSG_NOTIFY_FILE_CHANGE(
        MemHandle          data);
```

This notification is sent out whenever the file system changes in any way.

Source: GCN mechanism.

Destination: Object on the GCNSLT_FILE_SYSTEM GCN list.

Parameters: *data* Handle of a **FileChangeNotificationData** block.

Return: Nothing.

Structures:

◆Objects

```
typedef struct {
      PathName          FCND_pathname;
      DiskHandle        FCND_diskHandle;
      FileChangeType    FCND_changeType;
} FileChangeNotificationData;
typedef ByteEnum FileChangeType;
/* These flags may be combined using | and &:
      FCT_CREATE
      FCT_DELETE
      FCT_RENAME
      FCT_CONTENTS
      FCT_DISK_FORMAT */
```

■ MSG_NOTIFY_DRIVE_CHANGE

```
@importMessage MetaGCNMessages, void MSG_NOTIFY_DRIVE_CHANGE(
        GCNDriveChangeNotificationType   type,
        word                             driveNum);
```

This is sent to notify various system utilities that a drive has been created or destroyed or has changed ownership from one installable file system driver to another.

Note that during system initialization, the ownership of a drive may change several times as more-specialized drivers are loaded. This means the recipient should not be surprised if it's told a drive has been created that it thought already existed.

Source: The kernel issues this notification whenever a filesystem driver creates or destroys a drive it manages. This includes when a specialized filesystem driver takes control of an existing drive.

Destination: Any object that has added itself to the GCNSLT_FILE_SYSTEM GCN list. It is intended for system objects, such as the GenFileSelector.

Parameters: *type* **GCNDriveChangeNotificationType**.

driveNum Number of the affected drive.

Return: Nothing.

■ MSG_NOTIFY_APP_STARTED

```
@importMessage MetaGCNMessages, void MSG_NOTIFY_APP_STARTED();
```

This message is sent out when an application attaches to the UI.

Source: GCN Mechanism.

Destination: Any object on the GCNSLT_APPLICATION system GCN list.

Parameters: None.

Return: Nothing.

■ MSG_NOTIFY_APP_EXITED

```
@importMessage MetaGCNMessages, void MSG_NOTIFY_APP_EXITED();
```

1.1

This message is sent out when an application thread exits.

Source: GCN Mechanism.

Destination: Any object on the GCNSLT_APPLICATION system GCN list.

Parameters: None.

Return: Nothing.

■ MSG_NOTIFY_DATE_TIME_CHANGE

```
@importMessage MetaGCNMessages, void MSG_NOTIFY_DATE_TIME_CHANGE();
```

This message is sent out when the date or time changes—whenever the system comes back or the system time is altered (e.g. by the User in Preferences).

Source: GCN Mechanism.

Destination: Any object on the GCNSLT_DATE_TIME system GCN list.

Parameters: None.

Return: Nothing.

■ MSG_NOTIFY_USER_DICT_CHANGE

```
@importMessage MetaGCNMessages, void MSG_NOTIFY_USER_DICT_CHANGE(
                MemHandle           sendingSpellBox,
                MemHandle           userDictChanged);
```

This message is sent out when an application attaches to the UI.

Source: GCN Mechanism.

Destination: Any object on the GCNSLT_DICTIONARY system GCN list.

Parameters: *sendingSpellBox* Handle of SpellBox that sent out the notification.

userDictChanged Handle of user dictionary that changed.

Return: Nothing.

◆ Objects

■ MSG_NOTIFY_KEYBOARD_LAYOUT_CHANGE

```
@importMessage MetaGCNMessages, void MSG_NOTIFY_KEYBOARD_LAYOUT_CHANGE();
```

This message is sent out when the keyboard layout is changing. Usually this involves a change in status of the floating keyboard. When passing this event to **GCNListSend()**, you must be sure to pass the GCNLSF_FORCE_QUEUE flag. (Otherwise, if you have a **GenPenInputControl** running on the same thread, it may try to remove itself from the list while you are sending this message.)

Source: GCN Mechanism.

Destination: Any object on the GCNSLT_KEYBOARD_OBJECT system GCN list.

Parameters: None.

Return: Nothing.

■ MSG_NOTIFY_EXPRESS_MENU_CHANGE

```
@importMessage MetaGCNMessages, void MSG_NOTIFY_EXPRESS_MENU_CHANGE(
                GCNExpressMenuNotificationTypes type,
                optr                affectedField);
```

This message is sent to notify various system utilities that an express menu has been created or destroyed. The recipient receives the optr of the field to which the affected express menu belongs, as all access to the express menu is via messages sent to the field.

Source: The UI issues this notification whenever a GenField object creates or destroys its express menu.

Destination: Any object that has added itself to the GCNSLT_EXPRESS_MENU. GCN list. It is intended for system utilities, such as the print spooler or a task-switching driver, that need to add objects to each express menu in the system.

Parameters:

type	What happened to the field.
affectedField	Which field of the menu was affected. (This will not be the optr of the express menu itself.)

Return: Nothing.

Structures:

```
typedef enum {
      GCNEMNT_CREATED,
      GCNEMNT_DESTROYED
} GCNExpressMenuNotificationTypes;
```

■ MSG_PRINTER_INSTALLED_REMOVED

```
@importMessage MetaGCNMessages, void MSG_PRINTER_INSTALLED_REMOVED();
```

1.1

This message is sent whenever a printer is installed or removed. The recipient of this message might call **SpoolGetNumPrinters()** to determine if any printers or fax machines are currently installed.

Source: GCN Mechanism.

Destination: Any object on the GCNSLT_INSTALLED_PRINTERS system GCN list.

Parameters: None.

Return: Nothing.

■ MSG_META_CONFIRM_SHUTDOWN

```
@importMessage MetaGCNMessages, void MSG_META_CONFIRM_SHUTDOWN(
        GCNShutdownControlType type);
```

This message is sent out when the system is about to shut down.

All applications which need to keep the system from shutting down must add themselves to GCNSLT_SHUTDOWN_CONTROL and handle this message.

Source: The task switch mechanism, through GCN.

Destination: Any object on the GCNSLT_SHUTDOWN_CONTROL system GCN list.

Parameters: None.

Return: Nothing.

Interception: If the system is about to be suspended or shut down (if the passed **GCNShutDownControlType** is GCNSCT_SUSPEND or GCNSCT_SHUTDOWN), then any object receiving this message *must* call **SysShutdown()**, passing SST_CONFIRM_START before it puts up any dialog box it uses to ensure the user isn't doing something foolish. If **SysShutdown()** returns *true* (indicating something has already denied the shutdown request), the caller should *not* put up its confirmation box, nor need it call **SysShutdown()** again.
Once the object has received a response from the user, either affirmative or negative, it must call **SysShutdown()**, passing

SST_CONFIRM_ACK or SST_CONFIRM_DENY as appropriate. This will allow any other confirmations to happen, as well as sending the final result to the original caller of **SysShutdown()**.

If the passed control type is GCNSCT_UNSUSPEND, no response is required.

1.1.3.5 IACP Meta Messages

IACP is fully discussed in “Applications and Geodes,” Chapter 6 of the Concepts Book.

■ MSG_META_IACP_PROCESS_MESSAGE

```
@importMessage MetaIACPMessages, void MSG_META_IACP_PROCESS_MESSAGE(
        EventHandle         msgToSend,
        TravelOption        topt,
        EventHandle         completionMsg);
```

This message dispatches an IACP message to its proper destination, sending a completion message back when that has finished.

Source: **IACPSendMessage()**.

Destination: Any object registered as an IACP server, or the GenApplication object of a geode that is a client of such a server.

Parameters:

msgToSend	EventHandle of recorded message that the other side of the connection is actually sending.
topt	**TravelOption** (or -1 if *msgToSend* should be dispatched via **MessageDispatch() o**r delivered via MSG_META_SEND_TO_CLASSED_EVENT.
completionMsg	**EventHandle** of recorded message to send when the message in *msgToSend* has been handled. If null, then no completion message will be sent.

Interception: if you have an object other than the GenApplication object that is an IACP server, you will need to intercept this message. You do not want to pass it on to the superclass in this case; usually, you will just want to call **IACPProcessMessage()**.

■ MSG_META_IACP_NEW_CONNECTION

```
@importMessage MetaIACPMessages, void MSG_META_IACP_NEW_CONNECTION(
          MemHandle           appLaunchBlock,
          Boolean             justLaunched,
          IACPConnection      connection);
```

This message informs servers that a new client has connected to the server.

Source: **IACPConnect()**.

1.1

Destination: Any object registered as an IACP server.

Parameters: *appLaunchBlock* Handle of **AppLaunchBlock** passed to **IACPConnect()**. Do not free this block.

justLaunched *true* if the recipient was just launched (i.e. it received the **AppLaunchBlock** in its MSG_META_ATTACH call).

connection **IACPConnection** that is now open.

Interception: Must intercept if you want to do anything about receiving the new client; there is not default handler for this message. If you do not intercept this message, no harm is done.

■ MSG_META_IACP_LOST_CONNECTION

```
@importMessage MetaIACPMessages, void MSG_META_IACP_LOST_CONNECTION(
          IACPConnection      connection,
          word                serverNum);
```

This message informs a server (or client) that one of its clients (or servers) has shut down.

Source: **IACPShutdown()**.

Destination: Any object registered as an IACP server, or the GenApplication object of a geode who is a client of such.

Parameters: *connection* **IACPConnection** being closed.

serverNum Server number that shut down, or 0 if this was a client that shut down (and thus it is a server being notified through this message).

Interception: Must be intercepted to provide custom behavior upon losing a connection, as there is no default handler for this message. **IACPLostConnection()** is a good routine for servers to call to ensure that connections don't linger after a client has shut down its end.

◆ Objects

■ MSG_META_IACP_SHUTDOWN_CONNECTION

```
@importMessage MetaIACPMessages, void MSG_META_IACP_SHUTDOWN_CONNECTION(
         IACPConnection     connection);
```

This message shuts down the appropriate side of the indicated connection.

Source: **IACPLostConnection()**, though after a delay.

Destination: Any IACP server object.

Parameters: *connection* **IACPConnection** to shutdown.

Interception: Must be intercepted to finish the work of a call to **IACPLostConnection()**. Call **IACPShutdownConnection()** to get default handling of this message.

■ MSG_META_IACP_DOC_OPEN_ACK

```
@importMessage MetaIACPMessages, void MSG_META_IACP_DOC_OPEN_ACK(
         IACPDocOpenAckParams   *params);
```

This message is sent when a document has been opened; the document must have previously been passed in the **AppLaunchBlock** when the IACP connection was made. The optr of the GenDocument object managing the document is passed so that messages can be sent to it explicitly, though these messages must always be sent via IACP (with a **TravelOption** of -1) to allow the application to exit at any time.

Source: GenDocumentGroup.

Destination: IACP client (usually the GenApplication object of the client application).

Parameters: *params* Pointer to an **IACPDocOpenAckParams** structure.

Structures:

```
typedef struct {
      optr              IDOAP_docObj;
      IACPConnection    IDOAP_connection;
      word              IDOAP_serverNum;
} IACPDocOpenAckParams;
```

IDOAP_docObj stores the optr of the document object managing the document.

IDOAP_connection stores the **IACPConnection** over which the open request was received.

IDOAP_serverNum stores the server number of the GenApplication object acting as the document object's server, or zero if the connection is through some other object.

Interception: No default handler is defined. You must intercept this message to provide custom behavior.

1.1

■ MSG_META_IACP_DOC_CLOSE_ACK

```
@importMessage MetaIACPMessages, void MSG_META_IACP_DOC_CLOSE_ACK(
        IACPDocCloseAckParams      *params);
```

This message acts as the acknowledgment sent by a GenDocument object after it successfully processes MSG_GEN_DOCUMENT_CLOSE. Documents opened via IACP always operate in transparent mode; i.e. if you close a dirty file, it will be saved. If you don't want this behavior, you will have to send a message to revert the document.

Source: GenDocument object.

Destination: IACP client.

Parameters: *params* Pointer to a **IACPDocCloseAckParams** structure.

Structures:

```
typedef struct {
      optr              IDCAP_docObj;
      IACPConnection    IDCAP_connection;
      word              IDCAP_serverNum;
      word              IDCAP_status;
} IACPDocCloseAckParams;
```

IDCAP_docObj stores the optr of the document object that was managing the document.

IDCAP_connection stores the **IACPConnection** over which the close request was received.

IDCAP_serverNum stores the server number of the GenApplication object acting as the document object's server, or zero if the connection is through some other object.

◆Objects

IDCAP_status stores the **DocQuitStatus** of the close operation.

Interception: No default handler is defined. You must intercept this message to provide custom behavior.

1.2 ProcessClass

ProcessClass implements all the functionality for creating and managing the process aspect of a geode. It creates the process thread and associated message queues. **ProcessClass** is a subclass of **MetaClass** and is the superclass of **GenProcessClass**, below. You will probably not use **ProcessClass** directly, but you will almost certainly use **GenProcessClass**.

Those rare geodes which will use this class probably won't explicitly declare a Process object, but instead pass **ProcessClass** as an argument to that function in Appendix B of the Concepts manual.

ProcessClass has no instance data fields of its own. The messages defined by it are listed below:

■ MSG_PROCESS_NOTIFY_PROCESS_EXIT

```
void        MSG_PROCESS_NOTIFY_PROCESS_EXIT(
            GeodeHandle        exitProcess,
            word               exitCode);
```

This is sent to a Process object when one of its child processes exits. Many types of processes do not need to know when a child process exists; these processes need not handle this message.

Source: Kernel

Destination: Process of creating geode of child process exiting.

Parameters: This message is provided as notification only, i.e. there is no default handling of it. May be intercepted as desired.

Parameters: *exitProcess* Child process that exited.

exitCode Exit code. May be an error code.

Return: Nothing.

■ MSG_PROCESS_NOTIFY_THREAD_EXIT

```
void      MSG_PROCESS_NOTIFY_THREAD_EXIT(
          ThreadHandle        exitProcess,
          word                exitCode);
```

This message is sent to a Process object when a thread owned by it (via the **ThreadCreate()** routine) exits.

Source: Kernel.

1.2 **Destination:** Process owning thread which is exiting.

Interception: This message is provided as notification only (i.e. there is no default handling of it). May be intercepted as desired.

Parameters: *exitProcess* Handle of thread that exited.

exitCode Exit code (may be an error code).

Return: Nothing.

■ MSG_PROCESS_MEM_FULL

```
void      MSG_PROCESS_MEM_FULL(
          word   type);
```

This message is sent to a Process object by the memory manager when the heap is getting full. A Process object receiving this message should try to free memory (or mark it discardable) if possible.

Source: Kernel's heap manager.

Destination: All processes.

Interception: Any process which can adjust the amount of memory that it is using should respond to this message by reducing its demands on the system heap. For instance, buffers or UI object trees kept purely for performance reasons could be freed or reduced in number.

Parameters: *type* **HeapCongestion**.

Return: Nothing.

Structures:

```
typedef enum {
     /* HC_SCRUBBING:
      * Heap is being scrubbed. */
     HC_SCRUBBING,
```

Objects

```
            /* HC_CONGESTED:
             * Scrubber couldn't free a satisfactory
             * amount of memory. */
            HC_CONGESTED,
            /* HC_DESPERATE:
             * Heap is perilously close to overflowing. */
            HC_DESPERATE
      } HeapCongestion;
```

■ MSG_PROCESS_CREATE_UI_THREAD

```
Boolean  MSG_PROCESS_CREATE_UI_THREAD(
         ThreadHandle        *newThread,
         ClassStruct         *class,
         word                stackSize);
```

This is a low-level utility message requesting that the process create the UI thread for an application. Does nothing more than MSG_PROCESS_CREATE_EVENT_THREAD, but is split out to allow for interception or to change class or stack size.

Source: First thread of application process if geode has one or more resources marked as "ui-object" blocks.

Destination: First thread of application process.

Interception: May be intercepted to change class or stack size before calling superclass, or to replace default handling completely.

Parameters: *newThread* — *Pointer to a* **ThreadHandle** *buffer to store the created thread handle.*

class — Object class for the new thread. If you don't have any special messages to handle in this thread, besides those intended for objects run by the thread, you can just specify **ProcessClass** as the object class.

stackSize — Stack size for the new thread (3K bytes is probably reasonable).

Return: *true* if the thread was not created because of some problem.

Warnings: Be careful of deadlock situations.

■ MSG_PROCESS_CREATE_EVENT_THREAD

```
Boolean  MSG_PROCESS_CREATE_EVENT_THREAD(
         ThreadHandle        *newThread
         ClassStruct         *class,
         word                stackSize);
```

1.2

This message is a utility that creates a new event-driven thread owned by the recipient Process object. Typically, a Process object will send this message to itself when it needs an additional event thread. (It cannot be used to create a non-event driven thread. Use **ThreadCreate()** for this purpose instead.) This is implemented at ProcessClass and takes care of all the details of creating a new event-driven thread. The thread will always receive a MSG_META_ATTACH as its first event.

Source: Unrestricted.

Destination: Any process. This process will own the thread.

Interception: Not necessary, as the default handler provides the message's utility.

Parameters: *newThread* *Pointer to a* **ThreadHandle** *buffer to store the created thread handle.*

class Object class for the new thread. If you don't have any special messages to handle in this thread besides those intended for objects run by the thread, you can just specify **ProcessClass** as the object class.

stackSize Stack size for the new thread. 512 bytes is probably reasonable. If the thread will be running any objects that can undergo keyboard navigation (like dialog boxes and triggers and so forth), you'll probably want to make it 1K. The kernel already adds some extra space for handling interrupts (about 100 bytes).

Return: *true* if the thread was not created because of some problem.

Warnings: Be careful of deadlock situations.

◆ Objects

1.3 GenProcessClass

GenProcessClass is the class that you will use to define the Process object of your applications. This class includes some functionality for opening and closing geodes as well as saving to and restoring from state files. Typically, your application will define its own subclass of **GenProcessClass**; this subclass will be used as your Process object and will receive all messages destined for the Process.

In this subclass you can alter the **GenProcessClass**, **ProcessClass**, or **MetaClass** messages you need. More often, however, the subclass will be used to define new messages that are application-global or that should be handled by your Process object.

1.3.1 Starting and Stopping

For information about the steps involved in stopping, starting, or restoring an application (and to get context information about the messages described below), see "GEOS Programming," Chapter 5 of the Concepts Book.

Many of the following messages need AppAttachFlags to tell them how the process is attaching:

```
typedef WordFlags AppAttachFlags;
#define AAF_RESTORING_FROM_STATE    0x8000
#define AAF_STATE_FILE_PASSED       0x4000
#define AAF_DATA_FILE_PASSED        0x2000
#define AAF_RESTORING_FROM_QUIT     0x1000
```

- AAF_RESTORING_FROM_STATE indicates that the application is coming up from a previous state, either by re-launching using a state file, or re-entering application mode from engine mode in the same session. The UI trees will be in whatever state they were left in, which may be different than the statically declared UI tree, depending on what occurred in the application when the state file was written out (for example, a dialog box may have been on-screen). If this flag is *false*, the application is starting up fresh. If this flag is *true*, the UI does not call MSG_META_LOAD_OPTIONS and ignores any "ON_STARTUP" hints or attributes.

- AAF_STATE_FILE_PASSED indicates that the application is restoring from the state file passed in an **AppLaunchBlock**, presumably passed as an argument in the message also. If this flag is set, AAF_RESTORING_FROM_STATE will also be set.
- AAF_DATA_FILE_PASSED indicates that the passed **AppLaunchBlock** contains the name of a data file that should be opened.
- AAF_RESTORING_FROM_QUIT indicates that the application was in the process of quitting, reached engine mode, and is now being started back up into application mode. If set, will also be set, and we will be brought up in whatever state we originally entered engine mode.

MSG_GEN_PROCESS_RESTORE_FROM_STATE

```
void      MSG_GEN_PROCESS_RESTORE_FROM_STATE(
          AppAttachFlags      attachFlags,
          MemHandle           launchBlock,
          MemHandle           extraState);
```

This message is sent by the User Interface when an application is being loaded from a state file. This is sent to the process itself from MSG_META_ATTACH, whenever the application is being invoked as in MSG_GEN_PROCESS_RESTORE_FROM_STATE mode. Data passed is the same as that in MSG_META_ATTACH. The default handler fetches the application mode message, either MSG_GEN_PROCESS_OPEN_APPLICATION or MSG_GEN_PROCESS_OPEN_ENGINE, as saved in the application object, and sends that message to the process.

Note that the blocks passed need not be freed, as this is done by the caller upon return.

Source: Default **GenProcessClass** handler for MSG_META_ATTACH.

Destination: Self.

Interception: Intercepted generally only so that application can retrieve previously saved data out of the state block passed.

Parameters: *attachFlags* Flags with information about the state and data files.

launchBlock Handle of **AppLaunchBlock**, or zero if none. This block contains the name of any document file passed into the application on invocation.

extraState Handle of extra state block, or zero if none. This is the same block as returned from MSG_GEN_PROCESS_CLOSE_APPLICATION or MSG_GEN_PROCESS_CLOSE_ENGINE, in some previous MSG_META_DETACH. Process objects often use this extra block to save global variables to state files.

Return: Nothing.

■ MSG_GEN_PROCESS_OPEN_APPLICATION

```
void    MSG_GEN_PROCESS_OPEN_APPLICATION(
        AppAttachFlags      attachFlags,
        MemHandle           launchBlock,
        MemHandle           extraState);
```

This message is sent by the User Interface when an application is being loaded from its resource blocks. Applications will often intercept this message to set up certain things before being put on-screen. This is the handler in which, for example, you would register for certain notifications such as the quick-transfer notification.

This is sent to the process itself from MSG_META_ATTACH, whenever the application is being restored to application mode (as opposed to engine mode), or whenever it is being invoked as in application mode. Data passed is the same as that in MSG_META_ATTACH. The default handler sets the GenApplication object GS_USABLE, and brings the UI up on screen.

This message may be intercepted to open up any data file passed, before the UI for the application is actually set GS_USABLE. Note that the blocks passed need not be freed, as this is done by the caller upon return.

Source: **GenProcessClass** default handler for MSG_META_ATTACH only.

Destination: Same object.

Interception: Frequently intercepted by an application's own process class to find out when an application is first coming alive in the system. You *must* pass this message on to the superclass, or the application will never come up. Be aware that the entire UI tree for the application is the equivalent of not usable (~GS_USABLE) before the superclass is called, and is usable and up on screen visually after it is called. Thus, it is best to do non-UI related things, and changing of generic attributes and hints *before* calling the superclass. You must wait until *after* calling the superclass to perform any operations which require that objects be fully usable (e.g. bringing up a dialog box).

Parameters: *attachFlags* State information about the state and data files.

launchBlock Handle of **AppLaunchBlock**, or zero if none. This block contains the name of any document file passed into the application on invocation.

extraState Handle of extra state block, or zero if none. This is the same block as returned from MSG_GEN_PROCESS_CLOSE_APPLICATION or MSG_GEN_PROCESS_CLOSE_ENGINE, in some previous MSG_META_DETACH.
Is freed by caller—subclasses should not free the extra state block.

Return: Nothing.

■ MSG_GEN_PROCESS_CLOSE_APPLICATION

```
MemHandle MSG_GEN_PROCESS_CLOSE_APPLICATION();
```

This message is sent by the User Interface whenever the application is being shut down (during a detach) and it had been launched in application (as opposed to engine) mode.

Source: **GenProcessClass** default handler for MSG_META_DETACH only

Destination: Self

Interception: Convenient place for code that needs to be executed before application exits, for non-engine mode cases. Superclass must be called.

Parameters: None.

Return: Handle of block to save (or NULL for none). Process objects often save global variables to a state file in an extra block. This is the handle of that block.

■ MSG_GEN_PROCESS_OPEN_ENGINE

```
void      MSG_GEN_PROCESS_OPEN_ENGINE(
          AppAttachFlags      attachFlags,
          MemHandle           launchBlock);
```

This is sent to the process itself from MSG_META_ATTACH, whenever the application is being restored to engine mode, or whenever it is being invoked as in engine mode. Data passed is the same as that in MSG_META_ATTACH. There is no default handler.

This message may be intercepted to open up any data file passed, before any engine commands are delivered to the process. Note that the blocks passed need not be freed, as this is done by the caller upon return.

Source: **GenProcessClass** default handler for MSG_META_ATTACH only.

Destination: Self.

Interception: Generally unnecessary, though can be intercepted if notification of going into this mode is necessary.

Parameters:

attachFlags	State of state and data files.
launchBlock	Handle of **AppLaunchBlock**, or NULL if none. This block contains the name of any document file passed into the application on invocation.

Return: Nothing.

■ MSG_GEN_PROCESS_CLOSE_ENGINE

```
void      MSG_GEN_PROCESS_CLOSE_ENGINE();
```

This message is sent by the User Interface whenever the application is being shut down (during a detach) and it had been launched in “engine” mode.

Source: **GenProcessClass** default handler for MSG_META_DETACH only.

Destination: Self.

Interception: Convenient place for code that needs to be executed before the application exits, for engine mode cases. Superclass must be called.

■ MSG_GEN_PROCESS_CLOSE_CUSTOM

```
MemHandle MSG_GEN_PROCESS_CLOSE_CUSTOM();
```

This message is sent by the User Interface whenever the application is being shut down (during a detach) and it had been launched in some custom mode (not application or engine) that **GenProcessClass** doesn’t know about.

Source: Subclass of **GenProcessClass**.

Destination: Self.

Interception: Convenient place for code that needs to be executed before the application exits, for custom mode cases. Superclass must be called.

Parameters: None.

Return: Handle of block to save (or NULL for none).

■ MSG_GEN_PROCESS_ATTACH_TO_PASSED_STATE_FILE

```
MemHandle MSG_GEN_PROCESS_ATTACH_TO_STATE_FILE(
          AppAttachFlags      attachFlags,
          MemHandle           launchBlock);
```

This message is sent by the User Interface whenever the application is being attached to a state file. This message is sent when either restoring from state or detaching. May be subclassed to provide forced state behavior.

Source: **GenProcessClass** default handler for MSG_META_ATTACH.

Destination: Self.

Interception: May be intercepted to force use of a particular state file (by changing the name of the state file to use before calling superclass).

Parameters: *attachFlags* **AppAttachMode** (matches that in **AppLaunchBlock**).

launchBlock Block of structure **AppLaunchBlock**.

Return: Handle of extra block of state data (zero for none).

■ MSG_GEN_PROCESS_CREATE_NEW_STATE_FILE

```
word MSG_GEN_PROCESS_CREATE_NEW_STATE_FILE(
           MemHandle appInstanceReference);
```

This replaces MSG_GEN_PROCESS_ATTACHED_TO_PASSED_STATE_FILE if no state file had been specified in that message. This message's default handler will create a new state file and attach it normally. Can be subclassed to provide forced state file usage (e.g. use a different naming scheme).

Source: **GenProcessClass** default handler for MSG_META_DETACH, if not quitting.

Destination: Self.

Interception: May be intercepted to change name of state file to create.

Parameters: *appInstanceReference*

Block handle to block of structure **AppInstanceReference**.

Return: VM file handle (NULL if you want no state file).

◆Objects

■ MSG_GEN_PROCESS_DETACH_FROM_STATE_FILE

```
void        MSG_GEN_PROCESS_DETACH_FROM_STATE_FILE(
            MemHandle           extraState,
            word                appStates);
```

This message is sent by the User Interface when the application is detaching or quitting (may or may not be attached to a state file) and the detach is nearly complete.

Source: **GenProcessClass** default handler for MSG_META_DETACH, if not quitting.

Destination: Self.

Interception: Not generally done. Default behavior is what you want.

Parameters:

extraState Block handle of extra block to be saved (as returned from MSG_GEN_PROCESS_CLOSE_APPLICATION, MSG_GEN_PROCESS_CLOSE_ENGINE or MSG_GEN_PROCESS_CLOSE_CUSTOM). If the block is not transferred to the state file), it must be freed (if non-zero) by the handler for this message.

appStates **ApplicationStates** record with information about the application state.

Return: Nothing.

■ MSG_GEN_PROCESS_INSTALL_TOKEN

```
void        MSG_GEN_PROCESS_INSTALL_TOKEN();
```

This message is sent by the desktop to a process to get that process to install its token and moniker lists into the token database.

Source: Generally whatever geode launched this application in engine mode (e.g. GeoManager).

Destination: **GenProcessClass** object of any geode launched in engine mode.

Interception: May be intercepted to install additional tokens. Default behavior installs only application icon.

Parameters: None.

Return: Nothing.

■ MSG_GEN_PROCESS_GET_PARENT_FIELD

```
optr    MSG_GEN_PROCESS_GET_PARENT_FIELD();
```

This message is sent by process-libraries (such as the Spool Object Library) to find out which field object is its parent. This message will return the field of the first client of the library.

Source: Unrestricted.

Destination: **GenProcessClass** object.

Interception: Not necessary, as default behavior implements utility.

Parameters: Nothing.

Return: Parent field.

■ MSG_GEN_PROCESS_SEND_TO_APP_GCN_LIST

```
void    MSG_GEN_PROCESS_SEND_TO_APP_GCN_LIST(@stack
        word                sendFlags,
        EventHandle         event,
        MemHandle           block,
        word                manufListType,
        ManufacturerID      manufID);
```

This message sends the specified event to all the objects registered with the passed GCN list. This message should be subclassed by UI Controller objects.

The handler here merely sends the request on to GenApplication using MSG_META_GCN_LIST_SEND. Controllers should use this message, however, over direct communication with the application object, to ensure orderly updating of the list status event. One such failure case which is fixed is two target text objects, one run by the process thread in a view, the other a GenText run by the UI thread. If the GenText has the target and the user clicks quickly on view then GenText, the GenText may process both messages about the target being lost and gained before the process text object receives its gained and lost pair. If both objects sent MSG_META_GCN_LIST_SEND directly to the GenApplication object, the GenText's status would be wiped out by the subsequent reporting by the process text object. This problem is avoided by having both process and UI objects call here to pass status update info. This works because target changes start out ordered in the UI thread, and that order is passed on to the process thread in either of the two cases.

Source: Any object wishing to update an application GCN list. Don't use queue-order altering message flags such as MF_PLACE_IN_FRONT

when sending this message. As a convention must be established for the flag MF_FORCE_QUEUE in order to ensure orderly results, the convention in use is this: Don't pass it.
A typical call should use only the MF_STACK, MF_FIXUP_DS, and/or MF_FIXUP_ES flags, if needed.

Destination: **GenProcessClass** (application process) only.

Interception: Should not generally be intercepted, as **GenProcessClass** provides correct behavior. If intercepted and not sent onto superclass, event passed must be destroyed and data block reference count decremented, to avoid leaving obsolete data on the heap.

Parameters:

sendFlags	Flags to pass on to **GCNListSend()**.
event	Classed event to pass on to the list.
block	Handle of extra data block, if used (otherwise NULL). Blocks of this type must have a reference count, which may be initialized with **MemInitRefCount()** and be incremented for any new usage with **MemIncRefCount()**. Methods in which they are passed are considered such a new usage, and must have **MetaClass** handlers which call **MemDerefCount()**. Current messages supported are MSG_META_NOTIFY_WITH_DATA_BLOCK and MSG_NOTIFY_FILE_CHANGE.
manufListType	This may be a **GCNStandardListType** or any other word which acts as a GCN list ID.
manufID	Manufacturer ID, which helps identify the GCN list.

1.3.2 Undo Mechanism

The Undo mechanism is implemented at the GenProcess level; this allows the undo mechanism to be applicable to any application or process. In general, Undo allows a process, usually within an application, but possibly within a library, to reverse changes made in the state of other objects. GEOS allows an almost unlimited number of stored and reversible Undo actions; the practical limit is somewhere around 100 actions.

Undo actions are stored within undo *chains*. These chains allow queued actions to be undone in reverse order. Each element in an undo chain is made up of an **UndoActionStruct**. These structures are usually added with an **AddUndoActionStruct**. This structure has several elements:

```
typedef struct {
      UndoActionStruct        AUAS_data;
      optr                    AUAS_output;
      AddUndoActionFlags      AUAS_flags;
} AddUndoActionStruct
```

A chain of undo actions is stored for each object. If you want your object to recognize undo-able actions, you must add the undo actions yourself and intercept MSG_META_UNDO when those actions are played back. The object should be able to understand the data within the **UndoActionStruct** to perform the Undo action.

There are two **AddUndoActionFlags** which affect when and how undo notification occurs. If AUAF_NOTIFY_BEFORE_FREEING or AUAF_NOTIFY_IF_FREED_WITHOUT_BEING_PLAYED_BACK is set in the **AddUndoActionFlags**, it not only receives MSG_META_UNDO but also receives MSG_META_UNDO_FREEING_ACTION when the undo mechanism frees the action. You can check the flags in the **AddUndoActionStruct** passed with this message to decide what action to take.

The object wishing to register an action for undo sends the process MSG_GEN_PROCESS_UNDO_START_CHAIN. For each action in this undo chain (there may be multiple actions in a single chain) send MSG_GEN_PROCESS_UNDO_ADD_ACTION; pass this message the **AddUndoActionStruct** action of the action to add. Finally, send MSG_GEN_PROCESS_UNDO_END_CHAIN to mark the end of this undo chain. Undo chains may be nested within each other.

The messages following this section also describe supplemental behavior that you may find useful. In addition to these messages, **GenProcessClass** also provides the following routines:

GenProcessUndoGetFile() returns a file handle of a Huge Array or DB item to hold undo information. Use this routine to get a file to put such undo information into.

GenProcessUndoCheckIfIgnoring() allows an application or library to check whether an application is ignoring undo information; in this case, it can avoid creating unnecessary undo information.

■ MSG_GEN_PROCESS_UNDO_START_CHAIN

```
void        MSG_GEN_PROCESS_UNDO_START_CHAIN(@stack
            optr    title,
            optr    owner);
```

This message notifies the process of the start of an undo-able action. Note that all this message does is increment a count—a new undo chain is created when the count goes from zero to one. This allows a function to perform a number of undo-able actions and have them all grouped as a single undo-able action.

Source: Any object wanting to register an action for undo.

Destination: **GenProcessClass** only.

Interception: In general, should not be intercepted.

Parameters: *title* The null-terminated title of this action. If NULL, then the title of the undo action will be the title passed with the next UNDO_START_CHAIN.

owner The object which owns this action.

Return: Nothing.

■ MSG_GEN_PROCESS_UNDO_END_CHAIN

```
void        MSG_GEN_PROCESS_UNDO_END_CHAIN(
            BooleanWord         flushChainIfEmpty);
```

This message notifies the process of the end of an undo-able action. Note that all this message does is decrement a count—the current undo chain is terminated when the count goes from one to zero. This allows a function to perform a number of undo-able actions and have them all grouped as a single undo-able action.

Source: Any object wanting to register an action for undo.

Destination: **GenProcessClass** only.

Interception: In general, should not be intercepted.

Parameters: *flushChainIfEmpty*

Non-zero if you want to delete the chain if it has no actions; zero if the chain is OK without actions (actions will be added later).

Return: Nothing.

■ MSG_GEN_PROCESS_UNDO_ABORT_CHAIN

```
void        MSG_GEN_PROCESS_UNDO_ABORT_CHAIN();
```

This message aborts the current undo chain, destroying all actions in place on the current chain, and instructs the undo mechanism to ignore any undo information until the current undo chain is ended. This latter behavior is needed because the current chain may be nested within several chains, so we must ignore undo chain actions until the outermost chain is ended.

Source: Unrestricted.

Destination: GenProcess object.

Interception: Do not intercept.

■ MSG_GEN_PROCESS_UNDO_ADD_ACTION

```
VMChain     MSG_GEN_PROCESS_UNDO_ADD_ACTION(
            AddUndoActionStruct *data);
```

This message adds a new undo action to the current undo chain.

Source: Any object wanting to register an action for undo.

Destination: **GenProcessClass**.

Interception: In general, should not be intercepted.

Parameters: *data* Structure containing information which may be used to undo action.

Return: Will return NULL if we are ignoring undo messages.
If the value passed in *UAS_datatype* was UADT_PTR or UADT_VMCHAIN, then will return a VMChain or DBItem which may be used to undo the action. If neither of the above cases is true, return value is meaningless.

■ MSG_GEN_PROCESS_UNDO_GET_FILE

```
VMFileHandle MSG_GEN_PROCESS_UNDO_GET_FILE();
```

This message returns a VM file handle to store undo actions. This message is useful to access undo data in either a huge array or DB item. You may also use **GenProcessUndoGetFile()** to retrieve this file instead.

Source: Any object wanting to access the undo file.

Interception: Should not be intercepted.

Parameters: None.

Return: File handle of VM file with undo information.

■ MSG_GEN_PROCESS_UNDO_FLUSH_ACTIONS

```
void        MSG_GEN_PROCESS_UNDO_FLUSH_ACTIONS();
```

This message flushes the current undo chain (frees all undo actions, notifies edit control that there is no undo item).

Source: Any object using undo.

Interception: Should not be intercepted.

Parameters: None.

Return: Nothing.

■ MSG_GEN_PROCESS_UNDO_SET_CONTEXT

```
dword       MSG_GEN_PROCESS_UNDO_SET_CONTEXT(
            dword  context);
```

This message sets the current undo context. This allows the application to have separate undo chains associated with various documents or modes. This should be sent out before any other undo related messages. The document control automatically takes care of this when a document gets the model exclusive.

Passing NULL_UNDO_CONTEXT as the new context will trigger some zealous EC code if any other undo messages are sent while the context is null.

Source: Any object using undo.

Interception: Generally, should not be intercepted. Applications wanting to override the default behavior should at least flush out the current undo actions, as they will probably not be valid in the new context.

Parameters: *context* New context (this has no meaning to the undo mechanism—it's just a value).

Return: Old context.

Structures:

```
#define NULL_UNDO_CONTEXT 0
```

■ MSG_GEN_PROCESS_UNDO_GET_CONTEXT

```
dword     MSG_GEN_PROCESS_UNDO_GET_CONTEXT();
```

This message gets the current undo context.

Source: Any object using undo.

Interception: Generally, should not be intercepted.

Parameters: None.

1.3 **Return:** Current context.

■ MSG_GEN_PROCESS_UNDO_PLAYBACK_CHAIN

```
void      MSG_GEN_PROCESS_UNDO_PLAYBACK_CHAIN();
```

This message plays back the current undo chain, one action at a time. It simultaneously creates a “redo” chain, so the undone action can be redone.

Source: Edit control.

Interception: Generally, should not be intercepted.

Parameters: None.

Return: Nothing.

■ MSG_GEN_PROCESS_UNDO_IGNORE_ACTIONS

```
void      MSG_GEN_PROCESS_UNDO_IGNORE_ACTIONS(
          Boolean             flushActions);
```

This message causes a process to reject any undo messages.

Source: Edit control.

Parameters: *flushActions* *true* to flush the queue.

Interception: Generally, should not be intercepted.

■ MSG_GEN_PROCESS_UNDO_ACCEPT_ACTIONS

```
void      MSG_GEN_PROCESS_UNDO_ACCEPT_ACTIONS();
```

This message causes a process to accept undo messages again.

Source: Edit control.

Interception: Generally, should not be intercepted.

Objects

■ MSG_GEN_PROCESS_UNDO_CHECK_IF_IGNORING

```
Boolean  MSG_GEN_PROCESS_UNDO_CHECK_IF_IGNORING();
```

This message checks to see if the system is currently ignoring undo actions.

Source: Edit control.

Interception: Generally, should not be intercepted.

Parameters: None.

Return: Will return *true* if ignoring actions.

1.3

◆Objects

GenClass

2

GenClass provides the basic functionality, via message handlers and instance data, for all objects in the generic object library. *All* generic objects are subclassed off this master class. Therefore, all generic objects will contain common behavior in the instance fields and message handlers of **GenClass**. **GenClass** allows you to create generic trees, manage state data, and attach data such as visual monikers, keyboard accelerators, and even variable-length data to a specific object.

The Gen world, along with the Vis world, contains a large variety of pre-defined objects ready to install into your applications. **GenClass** itself is not a normally usable object, however. It should not be directly subclassed and used by your application. The functionality within this chapter will normally be accessed within one of **GenClass**' subclasses.

All of these pre-defined objects will contain behavior explained in this chapter. The instance data and message handlers described in this chapter may be implemented differently under different specific UIs, but the basic functionality will remain the same. Therefore, it is important to fully understand the concepts explained in this chapter before using any generic object.

You should be familiar with object-oriented programming and message handling before reading this chapter. See "The GEOS User Interface," Chapter 10 of the Concepts Book for an overview of each generic object in the user interface. You should be particularly familiar with setting up and modifying instance data.

2.1 GenClass Features

GenClass provides the instance fields and message handlers common to all generic objects. You will need to understand and manipulate **GenClass** operations in almost every generic object you create. **GenClass** is very powerful and diverse, providing a number of wide-ranging functions within your user interface. **GenClass** in specific provides the following:

- an abstract UI providing code common within any specific user interface.

- an object-oriented generic tree structure, including dynamic creation and destruction of generic UI objects “on the fly.”
- an object-oriented messaging system, including generic upward queries, allowing the UI objects to pass messages without defining a specific destination.
- pre-instantiation and persistence (state-saving) across sessions.
- window sizing and positioning with an abstract geometry manager without involved coordinate space computations.
- a generic visual update mechanism for your application, allowing you to ensure an accurate and visually up-to-date UI.
- the ability to create custom UI gadgets of your own.

Most importantly, **GenClass** provides the common functionality that will be accessed by all generic objects from the simplest GenTrigger to the most complex GenView.

2.2 GenClass Instance Data

Generic objects, along with visual objects, are the main means through which a user can communicate with an application. All generic objects follow an object-oriented class hierarchy with **GenClass** acting as their superclass. Therefore, all generic objects will share the attributes and behavior of **GenClass**, although many may not make use of the entire utility provided.

2.2.1 Instance Fields

GenClass provides a set of instance fields common to all subclasses. These instance fields are listed in Code Display 2-1.

Code Display 2-1 GenClass Instance Data Fields

```
    @instance @link             GI_link;
    @instance @visMoniker       GI_visMoniker;
    @instance @composite        GI_comp;
    @instance @kbdAccelerator   GI_kbdAccelerator;
    @instance GenAttrs          GI_attrs = 0;
    @instance GenStates GI_states = GS_USABLE | GS_ENABLED;

/* GenAttrs store attributes of a generic object. */

typedef ByteFlags GenAttrs;
#define GA_SIGNAL_INTERACTION_COMPLETE   0x80
#define GA_INITIATES_BUSY_STATE          0x40
#define GA_INITIATES_INPUT_HOLD_UP       0x20
#define GA_INITIATES_INPUT_IGNORE        0x10
#define GA_READ_ONLY                     0x08
#define GA_KBD_SEARCH_PATH               0x04
#define GA_TARGETABLE                    0x02
#define GA_NOTIFY_VISIBILITY             0x01

/* GenStates store the usability states of a generic object. */

typedef ByteFlags GenStates;
#define GS_USABLE       0x80
#define GS_ENABLED      0x40*
```

GI_visMoniker chooses the title for the generic object, usually added in an appropriate location within the object by the specific UI. The moniker can be text, in the form of a null-terminated text string, or as a graphics string, usually set up separately. For more information on visual monikers, see “Visual Monikers” on page 167.

GI_comp assigns child objects to the current object (the parent). *GI_comp* takes a list of generic objects separated by commas as arguments in its instance field. It then sets each of those arguments as children of the current object. In the Goc preprocessor, the *GI_comp* arguments will be used to create a generic tree structure, reassigning present links using both the *GI_comp* instance field and an internal instance field, *GI_link*. The *GI_link* instance field is therefore an elemental instance field for every generic object but is rarely, if ever, accessed directly. For more information, see “Composite Links” on page 181.

GI_kbdAccelerator sets a given keystroke as a keyboard accelerator for the object. Usually, an object is activated in a manner relating to its specific UI implementation. Pressing the accelerator keystroke will activate the object in the same manner as if its default activation action had occurred. The instance field takes a modifier (*alt, shift, ctrl,* or combinations of these) and a character (a valid keyboard character) as its arguments. You can further specify the specific keyboard set to use. For more information, see "Keyboard Accelerators" on page 184.

GI_attrs defines how actions performed by the generic object will affect user interface input and output. You may set any, all, or none of these attributes within your object declaration. For more information, see "Attributes" on page 186.

GI_states defines in what state a generic object will be represented. Default states are GS_USABLE and GS_ENABLED. You may set any, all, or none of these states within your object declaration. For more information, see "States" on page 191.

2.2.2 Vardata

GenClass provides a multitude of pre-defined vardata instance fields. You may add these vardata entries to a generic object directly within its object declaration. Vardata instance fields perform several key roles for a generic object. Specifically, they provide two distinct uses:

- hints to the specific UI on its implementation of an object. Hints always appear in the form HINT_*hintname*.
- an attribute data storage area considered part of an object's state data (which may therefore be saved to state). Vardata storage areas act as attributes attached to an object if they are explicitly declared within the object declaration. These data appear in the form ATTR_*attributename*. These data act in much the same way as fixed instance fields.

Any of these data may be accessed and altered using object system routines. For more information, see "GEOS Programming," Chapter 5 of the Concepts Book.

2.2.2.1 Vardata Attributes

Vardata attributes may contain the same functionality as other instance fields. Variable attributes, however, will only be allocated space within an object declaration if they are actually declared; instance data fields always take up space within an object, whether used or not. Therefore, vardata attributes are ideal for data that should only take up space if actually used. Vardata attributes are prefaced with ATTR_. Several ATTRs are included in **GenClass**. See Code Display 2-2 for a complete list of these vardata attributes.

Code Display 2-2 ATTR vardata fields

```
@vardata void           ATTR_GEN_PROPERTY;
@vardata void           ATTR_GEN_NOT_PROPERTY;

@vardata DestinationClassArgs    ATTR_GEN_DESTINATION_CLASS;
    @reloc ATTR_GEN_DESTINATION_CLASS, 0, optr;

@vardata char[]         ATTR_GEN_INIT_FILE_KEY;
@vardata char[]         ATTR_GEN_INIT_FILE_CATEGORY;
@vardata void           ATTR_GEN_INIT_FILE_PROPAGATE_TO_CHILDREN;
@vardata void           ATTR_GEN_USES_HIERARCHICAL_INIT_FILE_CATEGORY;

@vardata Point          ATTR_GEN_POSITION;
@vardata sword          ATTR_GEN_POSITION_X;
@vardata sword          ATTR_GEN_POSITION_Y;

@vardata void           ATTR_GEN_SEND_APPLY_MSG_ON_APPLY_EVEN_IF_NOT_MODIFIED;
@vardata void           ATTR_GEN_SEND_APPLY_MSG_ON_APPLY_EVEN_IF_NOT_ENABLED;

@vardata dword          ATTR_GEN_VISIBILITY_DATA;
@vardata word           ATTR_GEN_VISIBILITY_MESSAGE;
@vardata optr           ATTR_GEN_VISIBILITY_DESTINATION;
    @reloc ATTR_GEN_VISIBILITY_DESTINATION, 0, optr;

@vardata GenFilePath    ATTR_GEN_PATH_DATA;

@vardata ChunkHandle    ATTR_GEN_FEATURE_LINK;

@vardata GenDefaultMonikerType ATTR_GEN_DEFAULT_MONIKER;

@vardata optr           ATTR_GEN_OUTPUT_TRAVEL_START;
    @reloc ATTR_GEN_OUTPUT_TRAVEL_START, 0, optr;

/* Generic Help attributes */
```

```
@vardata char[]         ATTR_GEN_HELP_FILE;
@vardata byte           ATTR_GEN_HELP_TYPE;
@vardata void           ATTR_GEN_HELP_FILE_FROM_INIT_FILE;
@vardata optr           ATTR_GEN_FOCUS_HELP;
@vardata optr           ATTR_GEN_FOCUS_HELP_LIB;
    @reloc ATTR_GEN_FOCUS_HELP_LIB, 0, optr;
@vardata char[]         ATTR_GEN_HELP_CONTEXT;

/* Generic Window attributes */

@vardata MemHandle      ATTR_GEN_WINDOW_CUSTOM_LAYER_ID;
    @reloc ATTR_GEN_WINDOW_CUSTOM_LAYER_ID, 0, optr;
@vardata WinPriority    ATTR_GEN_WINDOW_CUSTOM_WINDOW_PRIORITY;
@vardata LayerPriority  ATTR_GEN_WINDOW_CUSTOM_LAYER_PRIORITY;
@vardata MemHandle      ATTR_GEN_WINDOW_CUSTOM_PARENT;
@vardata void           ATTR_GEN_CUSTOM_WINDOW;
@vardata void           ATTR_GEN_WINDOW_ACCEPT_INK_EVEN_IF_NOT_FOCUSED;
@vardata KeyboardOverride ATTR_GEN_WINDOW_KBD_OVERRIDE;
@vardata Point          ATTR_GEN_WINDOW_KDB_POSITION;
```

2.2

ATTR_GEN_PROPERTY and ATTR_GEN_NOT_PROPERTY affect a generic object's ability to behave as a properties group. These attributes override the property group's default behavior. Generic objects marked as properties in which case in delayed or immediate mode. For more information, see "Generic Properties" on page 235.

ATTR_GEN_DESTINATION_CLASS specifies an object class that should handle all messages sent out along a destination path. The type **DestinationClassArgs** contains a pointer to a **ClassStruct**. Typically, this attribute is used with a **TravelOption** destination to form a search path. The message for that generic object travels down a path specified in the **TravelOption** until it encounters an object class of type ATTR_GEN_DESTINATION_CLASS; at that point, the message will be handled. For more information, see "Destination Classes" on page 228.

ATTR_GEN_INIT_FILE_KEY, ATTR_GEN_INIT_FILE_CATEGORY, ATTR_GEN_INIT_FILE_PROPAGATE_TO_CHILDREN, and ATTR_GEN_USES_HIERARCHICAL_INIT_FILE_CATEGORY allow a generic object to load options from the .INI file upon startup (after receiving a MSG_META_ATTACH). Generic objects with these attributes will receive a MSG_GEN_LOAD_OPTIONS (with the category and key of the proper entry in the .INI file) and may intercept that message to provide additional behavior. For more information, see “Initialization File Management” on page 230.

ATTR_GEN_POSITION, ATTR_GEN_POSITION_X, and ATTR_GEN_POSITION_Y override default geometry management of an object and set a precise position for the object to appear within the UI, offset from the parent object. ATTR_GEN_POSITION specifies both a horizontal and a vertical position to place the object. ATTR_GEN_POSITION_X and ATTR_GEN_POSITION_Y specify either a horizontal or vertical position to align the object along. You should avoid using these attributes because they override default behavior and may result in different (and potentially unsightly) appearances under different specific UIs. For more information, see “Altering Default Geometry Management” on page 233.

ATTR_GEN_SEND_APPLY_MSG_ON_APPLY_EVEN_IF_NOT_MODIFIED and ATTR_GEN_SEND_APPLY_MSG_ON_APPLY_EVEN_IF_NOT_ENABLED affect generic objects that contain an apply message—a message that an object sends out when it wishes to apply changes made within that object. By default, an apply message will not be sent out if the object’s state has not changed since the last apply or if the object is not enabled; these attributes tell the object to send out its apply message regardless of these conditions. For more information, see “Altering Delayed Mode Activity” on page 234.

ATTR_GEN_VISIBILITY_DATA, ATTR_GEN_VISIBILITY_MESSAGE, and ATTR_GEN_VISIBILITY_DESTINATION set the data, message or destination to use with the visibility notification mechanism in **GenClass**. The visibility mechanism allows an application to know when an object becomes visible and not visible. An object provides this notification if its GA_NOTIFY_VISIBILITY bit is set. For more information, see “Notification of Visibility” on page 235.

ATTR_GEN_PATH_DATA stores an object’s generic path information in the form of a **GenFilePath** structure. This structure contains disk and path information including path name and file name; generic objects can use this attribute to store current or default paths and files. For more information, see “Generic Paths” on page 242.

ATTR_GEN_FEATURE_LINK contains a link to the next feature used by the GenControl mechanism when a controller’s features map to multiple objects. It is unlikely you will use this attribute. For more information, see “Feature Links” on page 242.

ATTR_GEN_DEFAULT_MONIKER specifies a default moniker for a generic object. This is generally used for graphic monikers that occur several places within the system. For more information, see “Default Monikers” on page 241.

ATTR_GEN_OUTPUT_TRAVEL_START allows you to set a specific object to begin the **TravelOption**. Normally, objects that contain a **TravelOption** in their destination field use their own object as the point of reference for sending messages along travel paths. (I.e sending a message TO_FOCUS would send a message along the focus path, starting at the sending object.) This can be used for optimization purposes, enabling you to skip several layers of hierarchies.

2.2

ATTR_GEN_HELP_FILE, ATTR_GEN_HELP_TYPE, ATTR_GEN_HELP_FILE_FROM_INIT_FILE, ATTR_GEN_FOCUS_HELP, ATTR_GEN_FOCUS_HELP_LIB, and ATTR_GEN_HELP_CONTEXT specify the configuration of the help files for a generic object. Focus help starts help notification when the generic object gains the focus. For more information on Help files, see “Help Management” on page 239.

ATTR_GEN_WINDOW_CUSTOM_LAYER_ID, ATTR_GEN_WINDOW_CUSTOM_WINDOW_PRIORITY, ATTR_GEN_WINDOW_CUSTOM_LAYER_PRIORITY, ATTR_GEN_WINDOW_CUSTOM_PARENT, and ATTR_GEN_CUSTOM_WINDOW specify a window’s interaction and place within the window system. ATTR_GEN_WINDOW_ACCEPT_INK_EVEN_IF_NOT_FOCUSED, ATTR_GEN_WINDOW_KBD_OVERRIDE, and ATTR_GEN_WINDOW_KBD_POSITION affect how input flows within a generic window. For more information, see “Window Management” on page 236.

2.2.2.2 Hints

Hints to the specific UI suggest added functionality to your application, and are prefaced with HINT_ (such as HINT_SEEK_REPLY_BAR). Hints may also have data structures attached to them. There are a large number of pre-defined hints with **GenClass**. See “Hints to the Specific UI” on page 244 for details on the operation of hints.

Many of the hints provided in **GenClass** manage the geometry of generic objects. For complete information on these hints, see “Managing UI Geometry,” Chapter 12 of the Concepts Book.

Code Display 2-3 Geometry Hints

```
/* These hints are explained within the Geometry Management chapter. */

@vardata void           HINT_DONT_ALLOW_CHILDREN_TO_WRAP;
@vardata void           HINT_ALLOW_CHILDREN_TO_WRAP;
@vardata word           HINT_WRAP_AFTER_CHILD_COUNT;
@vardata word           HINT_WRAP_AFTER_CHILD_COUNT_IF_VERTICAL_SCREEN;

@vardata void           HINT_BOTTOM_JUSTIFY_CHILDREN;
@vardata void           HINT_LEFT_JUSTIFY_CHILDREN;
@vardata void           HINT_RIGHT_JUSTIFY_CHILDREN;
@vardata void           HINT_TOP_JUSTIFY_CHILDREN;

@vardata void           HINT_FULL_JUSTIFY_CHILDREN_HORIZONTALLY;
@vardata void           HINT_FULL_JUSTIFY_CHILDREN_VERTICALLY;
@vardata void           HINT_DONT_FULL_JUSTIFY_CHILDREN;

@vardata SpecSizeSpec   HINT_CUSTOM_CHILD_SPACING;
@vardata SpecSizeSpec   HINT_CUSTOM_CHILD_SPACING_IF_LIMITED_SPACE;
@vardata void           HINT_DONT_INCLUDE_ENDS_IN_CHILD_SPACING;
@vardata void           HINT_INCLUDE_ENDS_IN_CHILD_SPACING;
@vardata void           HINT_MINIMIZE_CHILD_SPACING;

@vardata void           HINT_LEFT_JUSTIFY_MONIKERS;
@vardata void           HINT_CENTER_MONIKER;
@vardata void           HINT_DO_NOT_USE_MONIKER;
@vardata void           HINT_ALIGN_LEFT_MONIKER_EDGE_WITH_CHILD;

@vardata void           HINT_CENTER_CHILDREN_HORIZONTALLY;
@vardata void           HINT_CENTER_CHILDREN_ON_MONIKERS;
@vardata void           HINT_CENTER_CHILDREN_VERTICALLY;

@vardata optr           HINT_ALIGN_LEFT_EDGE_WITH_OBJECT;
    @reloc HINT_ALIGN_LEFT_EDGE_WITH_OBJECT, 0, optr;
@vardata optr           HINT_ALIGN_TOP_EDGE_WITH_OBJECT;
    @reloc HINT_ALIGN_TOP_EDGE_WITH_OBJECT, 0, optr;
@vardata optr           HINT_ALIGN_RIGHT_EDGE_WITH_OBJECT;
    @reloc HINT_ALIGN_RIGHT_EDGE_WITH_OBJECT, 0, optr;
@vardata optr           HINT_ALIGN_BOTTOM_EDGE_WITH_OBJECT;
    @reloc HINT_ALIGN_BOTTOM_EDGE_WITH_OBJECT, 0, optr;
```

```
@vardata CompSizeHintArgsHINT_FIXED_SIZE;
@vardata CompSizeHintArgsHINT_INITIAL_SIZE;
@vardata CompSizeHintArgsHINT_MAXIMUM_SIZE;
@vardata CompSizeHintArgsHINT_MINIMUM_SIZE;
@vardata void           HINT_SIZE_WINDOW_AS_DESIRED;

@vardata void           HINT_DRAW_IN_BOX;

@vardata void           HINT_EXPAND_HEIGHT_TO_FIT_PARENT;
@vardata void           HINT_EXPAND_WIDTH_TO_FIT_PARENT;

@vardata void           HINT_MAKE_REPLY_BAR;

@vardata void           HINT_NO_TALLER_THAN_CHILDREN_REQUIRE;
@vardata void           HINT_NO_WIDER_THAN_CHILDREN_REQUIRE;

@vardata void           HINT_ORIENT_CHILDREN_HORIZONTALLY;
@vardata void           HINT_ORIENT_CHILDREN_VERTICALLY;
@vardata void           HINT_ORIENT_CHILDREN_ALONG_LARGER_DIMENSION;
@vardata void           HINT_SAME_ORIENTATION_AS_PARENT;

@vardata void           HINT_PLACE_MONIKER_ABOVE;
@vardata void           HINT_PLACE_MONIKER_BELOW;
@vardata void           HINT_PLACE_MONIKER_TO_LEFT;
@vardata void           HINT_PLACE_MONIKER_TO_RIGHT;
@vardata void           HINT_PLACE_MONIKER_ALONG_LARGER_DIMENSION;
@vardata void           HINT_NO_BORDER_ON_MONIKERS

@vardata void           HINT_SEEK_MENU_BAR;
@vardata void           HINT_AVOID_MENU_BAR;

@vardata void           HINT_SEEK_REPLY_BAR;

@vardata void           HINT_SEEK_X_SCROLLER_AREA;
@vardata void           HINT_SEEK_Y_SCROLLER_AREA;

@vardata void           HINT_SEEK_LEFT_OF_VIEW;
@vardata void           HINT_SEEK_TOP_OF_VIEW;
@vardata void           HINT_SEEK_RIGHT_OF_VIEW;
@vardata void           HINT_SEEK_BOTTOM_OF_VIEW;

@vardata void           HINT_SEEK_TITLE_BAR_LEFT;
@vardata void           HINT_SEEK_TITLE_BAR_RIGHT;

@vardata void           HINT_USE_INITIAL_BOUNDS_WHEN_RESTORED;

@vardata void           HINT_DIVIDE_WIDTH_EQUALLY;
@vardata void           HINT_DIVIDE_HEIGHT_EQUALLY;
```

Objects

```
@vardata void            HINT_KEEP_INITIALLY_ONSCREEN;
@vardata void            HINT_DONT_KEEP_INITIALLY_ONSCREEN;
@vardata void            HINT_KEEP_PARTIALLY_ONSCREEN;
@vardata void            HINT_DONT_KEEP_PARTIALLY_ONSCREEN;
@vardata void            HINT_KEEP_ENTIRELY_ONSCREEN;
@vardata void            HINT_KEEP_ENTIRELY_ONSCREEN_WITH_MARGIN;

@vardata void            HINT_POPS_UP_TO_RIGHT;
@vardata void            HINT_POPS_UP_BELOW;

@vardata void            HINT_NOT_MOVABLE;

@vardata SpecWinSizePair HINT_POSITION_WINDOW_AT_RATIO_OF_PARENT;
@vardata void            HINT_POSITION_WINDOW_AT_MOUSE;

@vardata void            HINT_WINDOW_NO_TITLE_BAR;
@vardata void            HINT_WINDOW_NO_SYS_MENU;

@vardata void            HINT_STAGGER_WINDOW;
@vardata void            HINT_CENTER_WINDOW;
@vardata void            HINT_TILE_WINDOW;

@vardata void            HINT_EXTEND_WINDOW_TO_BOTTOM_RIGHT;
@vardata void            HINT_EXTEND_WINDOW_NEAR_BOTTOM_RIGHT;

@vardata SpecWinSizePair HINT_SIZE_WINDOW_AS_RATIO_OF_PARENT;
@vardata SpecWinSizePair HINT_SIZE_WINDOW_AS_RATIO_OF_FIELD;
@vardata void            HINT_WINDOW_NO_CONSTRAINTS;

@vardata SpecWinSizePair HINT_POSITION_ICON_AS_RATIO_OF_FIELD;
```

Other hints in **GenClass** do not affect geometry. These hints are listed below and are explained within this chapter. Within **GenClass**, some hints are internal and are not listed in this code display.

Code Display 2-4 Hints Explained Within This Chapter

```
/* These hints are explained within this chapter (or they are self-explanatory). */

@vardata void            HINT_FREQUENTLY_USED;
@vardata void            HINT_INFREQUENTLY_USED;
@vardata void            HINT_AN_ADVANCED_FEATURE;

@vardata void            HINT_DEFAULT_DEFAULT_ACTION;
@vardata void            HINT_ENSURE_TEMPORARY_DEFAULT;
@vardata void            HINT_PREVENT_DEFAULT_OVERRIDES;
```

```
@vardata void           HINT_SAME_CATEGORY_AS_PARENT;

@vardata void           HINT_USE_TEXT_MONIKER;
@vardata void           HINT_USE_ICONIC_MONIKER;

@vardata void           HINT_DEFAULT_FOCUS;
@vardata void           HINT_DEFAULT_TARGET;
@vardata void           HINT_DEFAULT_MODEL;

@vardata void           HINT_PRESERVE_FOCUS;
@vardata void           HINT_DO_NOT_PRESERVE_FOCUS;

@vardata void           HINT_GENERAL_CONSUMER_MODE;

@vardata void           HINT_NEVER_ADOPT_MENUS;
@vardata void           HINT_ALWAYS_ADOPT_MENUS;

@vardata void           HINT_NAVIGATION_ID;
@vardata void           HINT_NAVIGATION_NEXT_ID

@vardata void           HINT_DISMISS_WHEN_DISABLED;

@vardata void           HINT_HELP;

@vardata void           HINT_TOOLBOX;

@vardata void           HINT_SHOW_SHORTCUT;
@vardata void           HINT_DONT_SHOW_SHORTCUT;
@vardata void           HINT_DRAW_SHORTCUT_BELOW;

@vardata void           HINT_CAN_CLIP_MONIKER_WIDTH;
@vardata void           HINT_CAN_CLIP_MONIKER_HEIGHT;
@vardata void           HINT_SHOW_ENTIRE_MONIKER;

@vardata BackgroundColors        HINT_GADGET_BACKGROUND_COLORS;
@vardata word           HINT_GADGET_TEXT_COLOR;

@vardata SystemAttrs    HINT_IF_SYSTEM_ATTRS;
@vardata void           HINT_ENDIF;
@vardata void           HINT_ELSE;
```

HINT_FREQUENTLY_USED, HINT_INFREQUENTLY_USED, and HINT_AN_ADVANCED_FEATURE are descriptive hints that describe the generic object's role within the UI. It is, of course, left up to the specific UI to implement any special behavior based on these hints.

HINT_DEFAULT_DEFAULT_ACTION, HINT_ENSURE_TEMPORARY_DEFAULT, and

HINT_PREVENT_DEFAULT_OVERRIDES affect the default activation of objects within the UI. Objects with default activation can be activated indirectly (e.g., pressing the ENTER key within a dialog box activates a default trigger). For more information, see “Default Actions” on page 247.

HINT_SAME_CATEGORY_AS_PARENT indicates to the specific UI that this object should be treated in the same manner as its parent. Currently, this hint only affects GenInteractions. By default, GIV_SUB_GROUP GenInteractions within a GIV_POPUP menu have separator lines drawn around them to set them apart from other children. This hint removes those lines.

HINT_USE_TEXT_MONIKER and HINT_USE_ICONIC_MONIKER indicates that the generic object should choose either a text moniker or an iconic moniker if there exists a choice within the object’s moniker list. Currently, these hints are not implemented.

HINT_DEFAULT_FOCUS, HINT_DEFAULT_TARGET, and HINT_DEFAULT_MODEL specify that the generic object is the default focus, target, or model node at this level in the application. HINT_PRESERVE_FOCUS and HINT_DO_NOT_PRESERVE_FOCUS affect how the focus node changes after an object’s activation. For a complete discussion of the focus and target hierarchies, see “Input,” Chapter 11 of the Concepts Book.

HINT_GENERAL_CONSUMER_MODE instructs the specific UI to show the object in GCM mode. This hint takes an argument of type **GeneralConsumerModeFlags**; this record contains two three-bit flags: GCMF_LEFT_ICON and GCMF_RIGHT_ICON, which specify which icon to use on both the left and right sides of the title bar. Each icon must be of type **GCMIcon**; current types are GCMI_NONE, GCMI_EXIT, and GCMI_HELP.

HINT_NEVER_ADOPT_MENUS and HINT_ALWAYS_ADOPT_MENUS affect the behavior of menus placed within a GenDisplay. Menus can either sit within the GenDisplay or appear in the associated GenPrimary when that GenDisplay is the target. Menus within GenDisplays are discouraged, however, as they are a fairly advanced UI metaphor.

HINT_NAVIGATION_ID and HINT_NAVIGATION_NEXT_ID allow the field navigation path within a windowed object. HINT_NAVIGATION_NEXT_ID sets the object to navigate to from the current object. An object with a matching

HINT_NAVIGATION_ID must exist. For more information, see “Keyboard Navigation Hints” on page 248.

HINT_DISMISS_WHEN_DISABLED instructs the specific UI to dismiss the object when it is no longer fully enabled. This is typically useful for independently-displayable windows that should be dismissed in such cases.

HINT_HELP is currently unsupported.

HINT_TOOLBOX indicates that the generic object and all objects below it in the visual hierarchy are part of a “toolbox” and should draw themselves appropriately. Objects in toolboxes are typically much smaller than other components. For more information on toolboxes, see “GenInteraction,” Chapter 7.

HINT_SHOW_SHORTCUT, HINT_DONT_SHOW_SHORTCUT, and HINT_DRAW_SHORTCUT_BELOW affect the display of keyboard accelerators for generic objects. For more information, see “Keyboard Accelerators” on page 184.

HINT_CAN_CLIP_MONIKER_WIDTH and HINT_CAN_CLIP_MONIKER_HEIGHT instruct the specific UI to clip the object’s moniker height or width if not enough space is provided for them. Objects with HINT_SHOW_ENTIRE_MONIKER will force the object to leave enough space to show its entire moniker, if possible.

HINT_GADGET_BACKGROUND_COLORS causes grouping objects in toolboxes to draw custom background colors. The hint takes an argument of type **BackgroundColors**. Two colors can each be chosen for both the unselected state and the selected state; they will be blended together depending on the state of the object. If both colors are set to the same value, a solid color results. HINT_GADGET_TEXT_COLOR selects the text color of a visual moniker, typically within a toolbox. This hint takes an argument of type **TextColors**.

HINT_IF_SYSTEM_ATTRS, HINT_ENDIF, and HINT_ELSE may be used together to conditionally add hints to an object based on certain system criteria. If the **SystemAttrs** set in the HINT_IF_SYSTEM_ATTRS field are true for the current system, then the hints that follow (until a HINT_ENDIF is encountered) are included. If no HINT_ENDIF is encountered, then only the next hint is included. If the **SystemAttrs** do not match the current system, the following group of hints is deleted. For more information, see “System Attributes” on page 245.

Remember that these vardata entries are hints; they can be ignored by the specific UI and their implementation varies from object to object. Some hints, for example, are only relevant for certain object classes. Your mileage may vary.

2.3 GenClass Basics

In most cases, you will set much of your generic object’s instance data upon instantiation according to the values you include in your **.goc** file. You may also update and manipulate this instance data during execution with a variety of messages. The following section deals only with directly setting instance fields in your **.goc** file. For information on dynamically altering this instance data, see “Modifying GenClass Instance Data” on page 197.

2.3.1 Visual Monikers

A visual moniker is a text or graphics string attached to a generic object. This visual moniker displays the given string in a manner relevant to the specific implementation of an object. For example, a GenTrigger might create a button with text inserted as the object’s main visual implementation. The visual moniker in this case might cover the object’s total area. A GenPrimary, on the other hand, might only add such text in a title bar of the window. In all cases, the specific user interface has final control over how a visual moniker is displayed to the user.

2.3.1.1 Simple Visual Monikers

You may set the visual moniker of an object using the *GI_visMoniker* instance field. This instance field expects an argument of type **@visMoniker**. The visual moniker of an object is not actually stored within this instance field, however. The visual moniker of an object is stored within its own chunk, outside of the object itself. The *GI_visMoniker* instance field contains a chunk handle to this visual moniker chunk. Because a visual moniker can only be

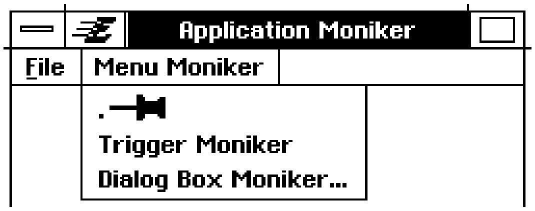


Figure 2-1 *Visual Monikers.*
The Title Bar of the Application contains a visual moniker, as do the menu, trigger and dialog box menu items.

referenced by a chunk handle, visual monikers for an object must reside in the same block as the object that points to them. The UI will set up this chunk and chunk handle automatically.

Within the object's definition, set the *GI_visMoniker* field to the desired visual moniker (either a text or graphics string). This can be done either by directly setting the *GI_visMoniker* field to the string or by indirectly setting the field to a visual moniker chunk defined elsewhere within your user interface with the Goc keyword **@visMoniker**. (See Code Display 2-5.)

Code Display 2-5 Setting Visual Monikers

```
/* A visual moniker can be set directly within the object's definition. */

@object GenTriggerClass MyTrigger = {
    GI_visMoniker = "Push Me";
}

/* A visual moniker may also be defined indirectly within a .goc file. */

/* The data for MyMoniker is set using the Goc keyword @visMoniker. This moniker
 * must reside in the same resource block as the object using it. This moniker
 * should be declared before being used in an object declaration. */

@visMoniker MyMoniker = "Push Me";
```

```
@object GenTriggerClass MyTrigger = {
    GI_visMoniker = @MyMoniker;
}
```

Remember that the *GI_visMoniker* instance field only stores a chunk handle to the actual string. The visual moniker for the object is not contained in the object chunk itself.

Text monikers may also specify a character to act as a mnemonic. Mnemonics are keyboard shortcuts used to activate an object without use of the mouse. Menus, menu items, and buttons frequently have mnemonic characters attached. This allows a user to navigate quickly to other UI objects or to activate them. If the specified character is within the text moniker, that letter will be highlighted in whatever fashion the specific UI decides is relevant. For instance, in OSF/Motif, mnemonic characters are underlined.

Usually the first letter of the text moniker is used for the mnemonic, though any character may be used. The specific UI will underline the first such occurrence of the character within the text moniker. If the character is not within the text itself, the specific UI may place the character within parentheses at the end of the text moniker. Mnemonics are activated according to specifications in the specific UI. In OSF/Motif, pressing the ALT key puts the user interface into its keyboard navigation mode. In this mode, all mnemonics for objects in the focus are activated by pressing the character of the mnemonic without need of an additional keystroke.

Mnemonics are set by enclosing the desired character in single quotes before the text string in your instance data. Mnemonics are case specific when defined. For example, if a visual moniker is the text “File,” only an uppercase mnemonic of F would highlight the first character in the text. Mnemonics are *not* case specific in their activation methods, however. For example, either *alt*-f or *alt*-F will activate a keyboard mnemonic of F.

Mnemonics are only valid for an object if that object is visually displayed and currently has the focus (see “Input,” Chapter 11 of the Concepts Book). Therefore, you may duplicate mnemonics for objects that will not exist at the same focus level. (An example of objects at the same focus level would be objects within the same menu.) Be careful not to duplicate the same

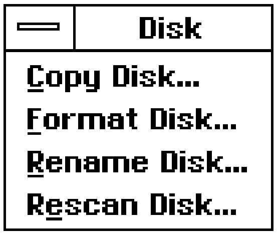


Figure 2-2 *Some examples of mnemonics.*
The characters C, F, R, and e are all mnemonics and are underlined in OSF/Motif.

mnemonic within the same focus level, as the UI will only associate the mnemonic with the first object that matches it.

The mnemonic may also store one of the following constants:

VMO_CANCEL corresponds to the specific UI defined cancel mnemonic. Use this rather than a special cancel mnemonic of your own choosing to keep the UI consistent.

VMO_MNEMONIC_NOT_IN_MKR_TEXT indicates that the mnemonic occurs after the null-terminator for the moniker text. This moniker is itself not null-terminated.

VMO_NO_MNEMONIC indicates that this moniker contains no mnemonic.

If you need other functionality that requires keyboard control of some form and mnemonics are not satisfactory, see “Keyboard Accelerators” on page 184.

Code Display 2-6 Setting Mnemonic Characters

```
/* Mnemonics are case specific. If the following example enclosed f instead of
 * F within single quotes, then the character f in parentheses would follow the
 * text “File.” Note that this case specificity does not apply to the user’s
 * activation of the object. That is, alt f will activate a mnemonic of F. */
```

Objects

```
@object MyTrigger GenTriggerClass {
    GI_visMoniker = ‘F’, “File”;
}

/* You can also specify the character to highlight with an actual numerical
 * position, counting from a zero-based (1st character is 0, 2nd character is 1,
 * etc.) character position in the text string. The “5” in the following example
 * will underline the F character (the sixth character in the text string.) */

    GI_visMoniker = 5, “Open File”;
```

2.3.1.2 Graphics String Visual Monikers

The *GI_visMoniker* instance field recognizes several keywords. These keywords are listed below and are usually only used in the construction of gstring visual monikers.

- style = { *text*, *abbrevText*, *graphicText*, *icon*, *tool* }
  This keyword specifies the **VMStyle** in use by this visual moniker.

text	Normal text moniker
abbrevText	Abbreviated text moniker (i.e., a short textual description rather than the full title). This is used for the name under an icon of an iconified GenPrimary.
graphicText	Textual gstring
icon	Normal gstring moniker
tool	Moniker for a tool, usually smaller than a standard visual moniker

- size = { *tiny*, *standard*, *large*, *huge* }
  This keyword specifies the **DisplaySize** that this moniker is intended for. It has nothing to do with the actual size of the moniker, which can be set with the keyword ‘cachedSize.’ Each size corresponds to a display type’s resolution level.

tiny	Tiny screens: CGA, 256x320.
standard	Standard screens: EGA, VGA, HGC, MCGA
large	Large screens: 800x680 SVGA
huge	Huge screens

You should not, in general, use monikers whose sizes are larger than their intended display type. (I.e. using ‘huge’ sized monikers on VGA screens.) You may use smaller monikers on larger screens, however.

- color = { *gray1*, *gray2*, *gray4*, *gray8*, *color2*, *color4*, *colorRGB* }
  This keyword specifies the color capability of the display that this setting is intended for. Each setting corresponds to a **DisplayClass**.

gray1	1 bit/pixel gray scale
gray2	2 bit/pixel gray scale
gray4	4 bit/pixel gray scale
gray8	8 bit/pixel gray scale
color2	2 bit/pixel color index
color4	4 bit/pixel color index
colorRGB	color with RGB values

- aspectRatio = { *normal*, *squished*, *verySquished* }
  This keyword specifies the aspect ratio of the display that this moniker is intended for.

normal	VGA, MCGA
squished	EGA, HGC (though normal is usually used)
verySquished	CGA

- cachedSize = { *X*, *Y* }
  This keyword sets the cached size (pixel by pixel) of the moniker.

- gstring { <*GString OpCodes*> }
  This keyword stores the gstring operations used in creating the visual moniker. These operations may also specify the creation of a bitmap by using the **Bitmap** operator.

Code Display 2-7 Examples of GString Visual Monikers

```
/* A Graphics string consisting of GString opcodes. */
    GI_visMoniker = {
        size = tiny;
        color = color4;
        aspectRatio = normal;
        cachedSize = 15,15;
        gstring{
            GSSaveTransform(),
            GSApplyRotation(45),
            GSFillEllipse(0,0,10,15),
            GSRestoreTransform(),
            GSDrawEllipse(0,0,10,15),
            GSEndString()
        }
    }

/* A Graphics string containing a bitmap. */
@visMoniker MyBitmap = {
        style = icon;
        size = standard;
        color = color4;
        aspectRatio = normal;
        cachedSize = 64, 40;
        gstring {
            GSDrawBitmapAtCP(166),
            Bitmap (64,40,BMC_PACKBITS, (BMT_MASK|BMF_4BIT)),
            251, 0,
            233, 221,
            ...,
            GSEndString()
        }
    }
```

2.3.1.3 Visual Moniker Lists

In some cases, an application may wish to use different visual monikers under specific circumstances. For example, a GenApplication's icon may need separate graphics strings for different display types (VGA, SVGA, etc.) In such cases, the argument for the moniker instance field should be given as a list of separate and distinct visual monikers. (See Code Display 2-8.)

You may then place each of these monikers within its own separate resource. Because only one moniker will be selected from the list, only one resource will be loaded into the object block, thereby conserving memory. When the application selects its appropriate moniker, the list will be replaced with the specific moniker, copied into the object block. The system performs this function automatically.

Code Display 2-8 Simple Lists

```
/* A list of monikers. */
    GI_visMoniker = list {
                        @moniker1,
                        @moniker2,
                        @moniker3
                }
```

2.3

```
/* If several monikers are specified in a list, they must be explicitly defined
 * somewhere else. If these monikers are complicated (as in the case of graphics
 * strings) they should each reside within their own resource because they will be
 * loaded in only once per application run. (Those that will be used together can
 * be within the same resource.) For example, in GEOS, several monikers
 * pertaining to different display types would be placed within separate resources.
 * When the appropriate moniker is selected, the list will be replaced with the
 * specific moniker. */

@start AppMonikerOneResource, notDetachable;

@visMoniker moniker1 = {
    size = large;
    color = color4;
    aspectRatio = normal;
    cachedSize = 64, 40;
    gstring {gstring data}
}

@end AppMonikerOneResource;

/* That moniker could then be declared within the object's instance data. */
@object GenPrimaryClass MyObject = {
    GI_visMoniker = list { @moniker1 }
}

/*
 * The GenApplication object usually contains a moniker list that allows the
 * specific UI to select a moniker based on the display.
 */
@object GenApplicationClass MyApplication = {
    GI_visMoniker = list {
        @TrigTextMoniker,       /* a simple text string */
        @TrigLCMoniker,         /* Large Color */
        @TrigLMMoniker,         /* Large Mono */
        @TrigSCMoniker,         /* Small Color */
        @TrigSMMoniker,         /* Small Mono */
```

◆ Objects

```
        @TrigLCGAMoniker,        /* Large CGA */
        @TrigSCGAMoniker         /* Small CGA */
    }
}

@visMoniker TrigTextMoniker = “Push Me”;

/* Graphics monikers might then appear within their own resource block. This
 * enables efficient memory management. */

@start AppMonikerResource, notDetachable;

@visMoniker TrigLCMoniker = {
    style = icon;
    size = large;
    color = color4;
    aspectRatio = normal;
    cachedSize = 64, 40;
    gstring {
        GSDrawBitmapAtCP(166),
        Bitmap (64,40,BMC_PACKBITS, (BMT_MASK|BMF_4BIT)),
        /*** insert Bitmap here ***/
        GSEndString()
    }
}

@end AppMonikerResource
```

2.3.1.4 The Inner Workings of Visual Monikers

The following section explains the inner workings of visual monikers within GEOS. It is not necessary to understand many of these concepts but it is illustrative of the system, and may aid in debugging and custom moniker use.

GI_visMoniker can take several types and combinations of arguments, all involving either text or graphics strings. Specifically, *GI_visMoniker* may indicate the following:

1) a simple text string
2) a text string with a mnemonic character
3) a graphics string consisting of GString opcodes

4) a graphics string containing a bitmap
5) a list of monikers of any type above.

Each of these arguments can be set up with the **@visMoniker** Goc keyword, but each will store their data in different manners. GEOS automatically sets up the visual moniker in the correct format.

All visual monikers make use of the **VisMoniker** structure. This basic structure indicates whether the moniker is text, a gstring, or a list of several types.

Code Display 2-9 The Basic VisMoniker Structure

```
/* The Basic VisMoniker structure contains a header which describes the type of
 * VisMoniker (VisMonikerType) and stores the cached width (width in pixels) of the
 * VisMoniker. The actual visual moniker data (either text or a gstring) follows
 * this header.
 *
 * If the visual moniker is text, this VisMoniker structure is contained within a
 * VisMonikerWithText structure. If the visual moniker is a gstring, this
 * VisMoniker structure is contained within a VisMonikerWithGString structure.*/

typedef struct {
    byte        VM_type;        /* VisMonikerType */
    word        VM_width;       /* Cached width of moniker */
} VisMoniker;

/* VisMonikerType specifies the type of moniker contained in the VisMoniker
 * structure.
 *
 * The flag VMT_MONIKER_LIST is actually a dummy flag. (This flag is never set
 * within a VisMoniker structure.) VisMonikerListEntryType has a matching flag
 * in the same location (VMLET_MONIKER_LIST). If that flag is set, it tells the
 * system that this isn’t actually a VisMoniker structure but is instead a
 * VisMonikerListEntry. This is used in moniker lists (see below).
 *
 * The flag VMT_GSTRING is set if the visual moniker is in the form of a gstring
 * instead of a simple text string. If this flag is set, VMT_GS_ASPECT_RATIO and
 * VMT_GS_COLOR specify the DisplayAspectRatio and DisplayClass used by this
 * gstring.
 */
```

```
typedef ByteFlags VisMonikerType;
#define VMT_MONIKER_LIST        0x80
#define VMT_GSTRING             0x40
#define VMT_GS_ASPECT_RATIO     0x30    /* DisplayAspectRatio */
#define VMT_GS_COLOR            0x0f    /* Color */

#define VMT_GS_ASPECT_RATIO_OFFSET 4
#define VMT_GS_COLOR_OFFSET 0
```

If the visual moniker is a simple text string, the ChunkHandle within *GI_visMoniker* will point to a chunk containing a **VisMonikerWithText** structure. This chunk will contain the basic **VisMoniker** header, along with the moniker's mnemonic character and a null-terminated text-string.

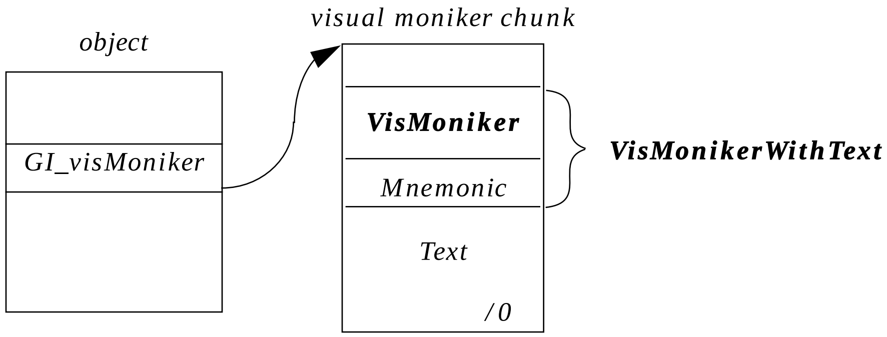


Figure 2-3 *A Simple Text Moniker*

The object and the visual moniker chunk reside in the same resource block. The null-terminated text appears after the ***VisMonikerWithText*** *structure.*

Code Display 2-10 VisMonikers With Text

```
/* If the VisMoniker contains simple text, the ChunkHandle within GI_visMoniker
 * points to a VisMonikerWithText structure. This structure contains the
 * basic VisMoniker header and the character of the mnemonic. (A value of -1 is
 * stored in VMWT_mnemonicOffset if there is no mnemonic for this visual moniker.)
 */
```

```
typedef struct {
    VisMoniker  VMWT_common;
    char        VMWT_mnemonicOffset;
} VisMonikerWithText;

/* The text, in the form of a null-terminated text string, follows this structure.
 * This text may be accessed (though it is not recommended) with the VMWT_text
 * offset. */

#define VMWT_text(sizeof(VisMonikerWithText)) /* Start of text. */
```

If the visual moniker is a gstring, the ChunkHandle within *GI_visMoniker* will point to a **VisMonikerWithGString** structure instead. This structure will contain the basic **VisMoniker** header, along with the moniker's cached height and the actual gstring. (The cached width is stored within the **VisMoniker** header.)

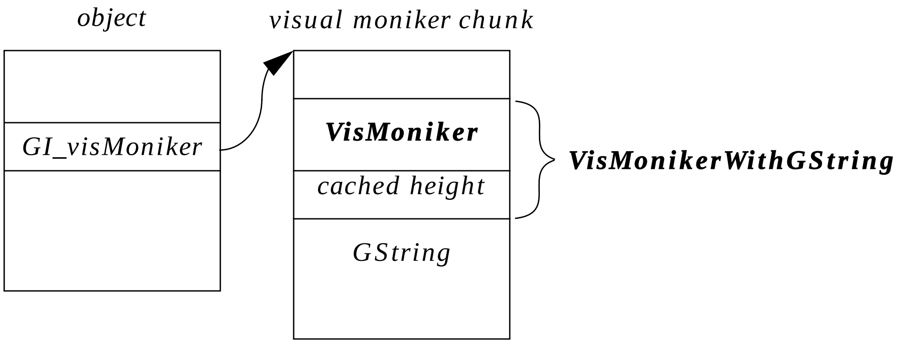


Figure 2-4 *A GString Moniker*

The object and the visual moniker chunk reside in the same resource block. The gstring appears after the ***VisMonikerWithGString*** *structure.*

Code Display 2-11 VisMonikers With GStrings

```
/* If the VisMoniker contains a gstring, the ChunkHandle within GI_visMoniker
 * points to a VisMonikerWithGString structure. This structure contains the
 * basic VisMoniker header and the cached height of the gstring. (The cached width
 * is stored within the VisMoniker structure.
 */
```

```
typedef struct {
    VisMoniker  VMWGS_common;
    word        VMWGS_height;
} VisMonikerWithGSTring;

/* The gstring follows this structure.This gstring may be accessed (though it is
 * not recommended) with the VMWGS_gString offset. */

#define VMWGS_gString(sizeof(VisMonikerWithGString)) /* Start of gstring. */
```

If instead of a single visual moniker, be it a text string or a gstring, *GI_visMoniker* specifies a list of monikers, the case is more complex.

If *GI_visMoniker* contains a list of monikers, that ChunkHandle will point to a group of **VisMonikerListEntry** structures (one for each moniker in the list). Each of these list entries will contain the type of moniker it references and the optr of the moniker it refers to. The actual moniker itself is not stored in that chunk. Also, because the moniker may be referred to by an optr, the actual visual monikers may reside in separate resources.

When the object containing the moniker list is first built, the system will select one of the monikers in the list (based on matching criteria in the *VMLE_type* field) and replace the moniker list with the single selected visual moniker.

Code Display 2-12 VisMoniker Lists

```
/* If GI_visMoniker contains a list of monikers instead of a single moniker, the
 * ChunkHandle of that instance field actually points to a collection of
 * VisMonikerListEntry structures. (The total number can be calculated by dividing
 * the size of the chunk by sizeof(VisMonikerListEntry) if needed.)
 *
 * Each VisMonikerListEntry structure contains a header which describes the type of
 * VisMoniker stored in that list entry (VisMonikerListEntryType) and the optr of
 * the stored moniker. The actual moniker may reside in a different resource. */

typedef struct {
    word        VMLE_type;
    optr        VMLE_moniker;
} VisMonikerListEntry;
```

2.3

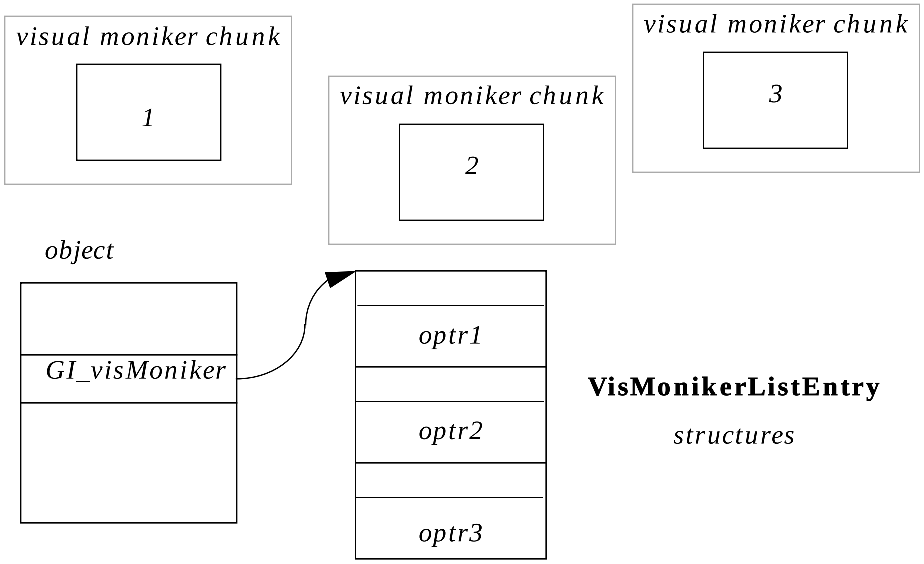


Figure 2-5 *A Visual Moniker List*

The GI_visMoniker instance field points to a chunk containing a number of VisMonikerListEntry structures. Each of these structures contains an optr to a visual moniker chunk, which may reside outside the current object block.

```
/* VisMonikerListEntryType specifies the type of moniker specified in the
 * VisMonikerListEntry structure.
 *
 * The flag VMLET_MONIKER_LIST must be set within this structure. This flag tells
 * the system that this isn’t actually a VisMoniker structure but is instead a
 * VisMonikerListEntry.
 *
 * The system uses the other flags in this structure to determine the type of
 * moniker contained as the list entry. The system will use this information to
 * select the most appropriate moniker that will satisfy the system’s needs. This
 * method is used most often in the selection of a GenPrimary’s main application
 * moniker based on the DisplaySize of the system.
 *
 * VMLET_GS_SIZE stores the DisplaySize that this moniker is most appropriate for.
 * This DisplaySize is set using the “size” entry within the visual moniker
 * declaration.
 *
 * VMLET_STYLE stores the VMStyle that this moniker most closely matches. This
 * VMStyle is set using the “style” entry within the visual moniker declaration.
 *
 * VMLET_GSTRING specifies that this moniker list entry is in the form of a gstring
 * If this flag is set, VMLET_GS_ASPECT_RATIO and VMLET_GS_COLOR specify the
 * DisplayAspectRatio and DisplayClass used by this gstring.
 */

typedef ByteFlags VisMonikerListEntryType;
#define VMLET_GS_SIZE            0x0300  /* DisplaySize. */
#define VMLET_STYLE              0x0f00  /* VMStyle */
#define VMLET_MONIKER_LIST       0x0080
#define VMLET_GSTRING            0x0040
#define VMLET_GS_ASPECT_RATIO    0x0030  /* DisplayAspectRatio */
#define VMLET_GS_COLOR           0x000f  /* DisplayClass */

#define VMLET_GS_SIZE_OFFSET12
#define VMLET_STYLE_OFFSET 8
#define VMLET_GS_ASPECT_RATIO_OFFSET 4
#define VMLET_GS_COLOR_OFFSET 0

/* VMStyle specifies the style of the visual moniker. The system may select a
 * moniker based on the style it wishes to display. */

typedef ByteEnum VMStyle;
#define VMS_TEXT 0               /* Simple text */
#define VMS_ABBREV_TEXT 1        /* Abbreviated text */
#define VMS_GRAPHIC_TEXT 2       /* Textual graphics string */
#define VMS_ICON 3               /* Normal gstring */
#define VMS_TOOL 4               /* Tool-sized gstring */
```

For information on manipulating visual monikers dynamically using **GenClass** messages, see “Managing Visual Monikers” on page 198.

2.3.2 Composite Links

Composite links form the connections between parent objects and child objects within a generic tree (see “The GEOS User Interface,” Chapter 10 of the Concepts Book). These links are set up using the *GI_comp* and *GI_link* instance fields. The *GI_comp* field points to an object’s first child. The *GI_link* field points to an object’s next sibling or to its parent if no next sibling exists. (See Figure 2-6 for an illustration.) In Goc, however, this usage is greatly simplified. The developer only needs to set the *GI_comp* field equal to its complete list of children for the parent object. The Goc preprocessor will then create and reassign all necessary links.

Code Display 2-13 Using GI_comp to Add Children

```
/* The GenInteraction (MyInteraction) acts as the parent object for the three
 * child GenTriggers. All three GenTrigger children will be placed within the
 * GenInteraction object. */

@object GenInteractionClass MyInteraction = {
    GI_visMoniker = “Menu”;                       /* Text Moniker */
    GII_visibility = GIV_POPUP;                   /* Creates a Menu */
    GI_comp = @MyFirstChild, @MySecondChild, @MyThirdChild;
                                                 /* list of children */
}

@object GenTriggerClass MyFirstChild= {
    GI_visMoniker = “Child 1”;           /* Text Moniker */
}

@object GenTriggerClass MySecondChild = {
    GI_visMoniker = “Child 2”;           /* Text Moniker */
}

@object GenTriggerClass MyThirdChild = {
    GI_visMoniker = “Child 3”;           /* Text Moniker */
}
```

This simple functionality is all you need to know to add children to your generic objects (and thus to create generic trees). However, it is somewhat helpful in certain cases (as in debugging) to understand what takes place underneath the surface. When an object in GEOS is assigned children, the preprocessor actually only assigns one composite link (*GI_comp*) to the first child. Each additional child acquires a link from its previous sibling using the internal instance field *GI_link*. Therefore, the parent will have a *GI_comp* to its first child, the first child will have a *GI_link* to the next sibling (the parent’s second child) and so forth. The last sibling (the parent’s last child) will have a *GI_link* back to the parent. This *GI_link* will have the LF_IS_PARENT bit set to indicate that the child points to a parent and not to a sibling. This forms what amounts to a circular linked list rather than a branching tree structure. (See Figure 2-6.)

This structure provides a simple and convenient usage. Any object will always have at most two links to other children or parent objects. Therefore the two instance fields *GI_comp* and *GI_link* provide the entire means of constructing a generic tree.

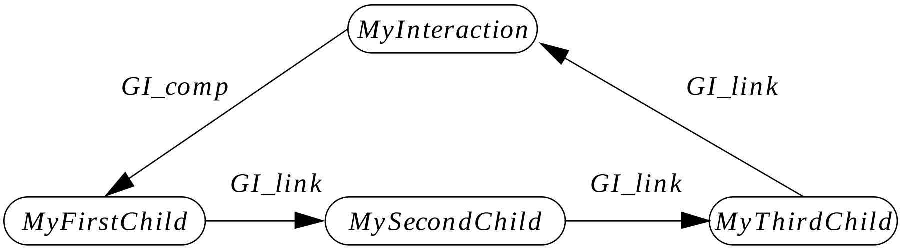


Figure 2-6 *A sample generic tree structure*
The GI_comp points to the first child. The GI_link points to the next sibling or to the parent object if no next sibling exists.

As can be seen in Figure 2-6, you can travel anywhere in the generic tree through these two links. For example, for MyInteraction to communicate with MyThirdChild, it follows the path of the *GI_comp* to the first child, MyFirstChild, and then continues through the two *GI_link*s to the third child. Conversely, a child can reach a parent by travelling along the *GI_link*s of siblings until it reaches the last sibling, whose *GI_link* points to its parent object.

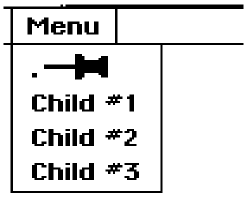


Figure 2-7 *A Menu with Three Children.*
A GenInteraction menu with three GenTrigger children.

GenClass message handlers provide several means of pointing to the proper parent/child object without needing to explicitly state the proper *GI_comp* and *GI_link* paths. In practice, you will never need to, and in fact should not, manipulate the *GI_link* field at all. This information is provided merely for your use in debugging your applications. In fact, you can usually assume that a conventional branching tree structure exists rather than the linked-list tree structure shown here.

For information on manipulating these links dynamically using **GenClass** messages, see "Generic Trees" on page 212.

2.3.3 Keyboard Accelerators

Keyboard accelerators permit the user to activate objects without the use of default activations in the specific UI. (Much of the default activations in most graphical user interfaces use the mouse, for example.) In most cases, these keyboard accelerators require the use of a modifier (the *alt*, *shift* or *ctrl* key) and another key. The keyboard accelerator may also be a function key (F10, etc.). In general, you should avoid using the *alt* key because of its special role as a keyboard mnemonic within a visual moniker (see "Visual Monikers" on page 167).

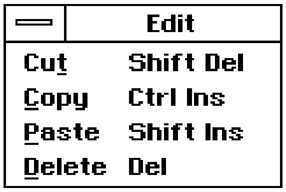


Figure 2-8 *Some Sample Keyboard Accelerators*
The keyboard accelerators are placed in the right-hand column of the menu. Notice that some entries have both a mnemonic and a keyboard accelerator.

Keyboard accelerators may be activated at any time; they are valid at any point within the application whether the generic object involved is being displayed currently or not. (The only exception to this is during the display of a modal dialog box, which will block input to any other part of the application.) The system will search through the generic tree for the object containing the proper key sequence. In this manner, accelerators differ from mnemonics, which are only valid for an active level. Therefore, only use keyboard accelerators for heavily utilized commands; this prevents a glut of modified keystrokes, which may confuse the user. In general, it is good practice to use the “control” key exclusively for keyboard accelerators to prevent overlapping with GEOS keyboard acclelators and mnemonics.

Table 2-1 *Valid Action Keys for Keyboard Accelerators*

These keys are only valid using the specific UI keyword. These Action Keys must also be combined with valid Modifiers (alt, shift, or ctrl).

NUMPAD_0	SPACE	UP
NUMPAD_1	TAB	DOWN
NUMPAD_2	ESCAPE	RIGHT
NUMPAD_3	F1	LEFT
NUMPAD_4	F2	HOME
NUMPAD_5	F3	END
NUMPAD_6	F4	PAGEUP
NUMPAD_7	F5	PAGEDOWN
NUMPAD_8	F6	INSERT
NUMPAD_9	F7	DELETE
NUMPAD_PLUS	F8	BACKSPACE
NUMPAD_MINUS	F9	MINUS
NUMPAD_DIV	F10	ENTER
NUMPAD_MULT	F11	
NUMPAD_PERIOD	F12	

You can set up keyboard accelerators using the *GI_kbdAccelerator* attribute. A keyboard accelerator must contain a valid modifier such as *alt*, *shift*, *control*, or *ctrl*. You may use any alpha-numeric character as the action key for the modifier to act on. In addition to alpha-numeric characters, the action keys within Table 2-1 are also available.

The specific UI has final control over how and whether a keyboard accelerator will be implemented. Currently, keyboard accelerators will only work for GenTriggers and GenItems. In most cases, a keyboard accelerator will be displayed alongside the text for the visual moniker of an object. In OSF/Motif, keyboard accelerators are shown to the right of menu items and to the left of other items (see Figure 2-8 on page ◆ 184). You may add HINT_DRAW_SHORTCUT_BELOW on non-menu items to draw the keyboard accelerator below the object’s visual moniker.

Code Display 2-14 Using a Keyboard Accelerator

```
@object GenTriggerClass MyTrigger = {
        /* Case is ignored for keyboard accelerators. */
    GI_kbdAccelerator = ctrl ‘R’;       /* ‘ctrl’ and ‘control’ are both valid. */
}

/ * Other examples:
    GI_kbdAccelerator = control ‘A’;

   Case is ignored for keyboard accelerators (unlike mnemonics). If you need
   an uppercase ‘A’, use ‘shift’ as a modifier.

    GI_kbdAccelerator = control shift ‘A’;
    GI_kbdAccelerator = alt ‘z’;
    GI_kbdAccelerator = alt shift ‘Z’;
    GI_kbdAccelerator = ctrl shift ‘G’;

   If you know what specific keyboard set will be implemented, you can add such
   special characters with the @specificUI keyword, though this is discouraged.

    GI_kbdAccelerator = @specificUI alt shift DELETE;
    GI_kbdAccelerator = @specificUI F6;          */
```

For information on manipulating keyboard accelerators dynamically, see “Managing Keyboard Accelerators” on page 210.

2.3.4 Attributes

GI_attrs, MSG_GEN_GET_ATTRIBUTES, MSG_GEN_SET_ATTRS

The *GI_attrs* attribute is a record specifying how an object will behave under various circumstances. In some cases, the *GI_attrs* instance field indicates that an object may initiate a busy state within the application. Several attributes only affect how an object will behave during activation. Prior to an object’s activation, some of these attributes will not affect the object’s behavior.

GA_SIGNAL_INTERACTION_COMPLETE
This flag instructs the UI that an Interaction has been completed. The specific UI may then decide whether to dismiss the Interaction or not. Most often, this attribute is attached to

a GenTrigger within an independently displayable GenInteraction (dialog box). This attribute is useful for dismissing temporary dialogs that request information from the user. This attribute will cause the generic object to send a MSG_GEN_GUP_INTERACTION_COMMAND to its generic parent with the **InteractionCommand** IC_INTERACTION_COMPLETE. This message is then passed up the generic tree until it reaches an appropriate Interaction. See "GenInteraction," Chapter 7 for more information.

GA_INITIATES_BUSY_STATE

This flag instructs the UI to mark the application as busy whenever the generic object is activated. You should only set this attribute for objects that may initiate long operations. This attribute requests that the UI visually change the cursor to show that the application is busy. (In OSF/Motif this is represented by an hourglass.) When the application finishes its operation, the cursor will revert to its former state. This attribute causes the object to send MSG_GEN_APPLICATION_MARK_BUSY to the object's application object along with MSG_GEN_APPLICATION_MARK_NOT_BUSY sent via the process' event queue. This allows the system to process the first message (showing the busy cursor) and only process the second message (removing the busy cursor) after the application finishes its current operation. (If the action is processed quickly, the cursor will often not appear.) The busy state cursor is only reflected in the current application; if the cursor roams outside the application bounds, the default behavior will occur.

GA_INITIATES_INPUT_HOLD_UP

This flag instructs the UI to mark the application as busy and to delay processing input messages until the UI and Application queues have been flushed. The application will then complete its current operation before beginning to process its input events. You should set this attribute when an object's activation may modify the UI, thereby preventing the user from clicking on objects that may become invalid after the operation. This attribute immediately sends a MSG_GEN_APPLICATION_HOLD_UP_INPUT to the application object along with a MSG_GEN_APPLICATION_RESUME_INPUT delayed via the process' event queue. This functionality is only reflected in the current application; if the cursor roams outside the application bounds, the default behavior will occur.

GA_INITIATES_INPUT_IGNORE

This flag instructs the UI to mark the application as busy and to completely ignore all subsequent input events to the application. The application will enter a *modal* state, meaning that all other application events will be ignored until the UI and Application queues are flushed. Usually, this state is conveyed to the user by broadcasting an audible beep whenever an input event is attempted. The generic object will immediately send a MSG_GEN_APPLICATION_IGNORE_INPUT to the application object along with a MSG_GEN_APPLICATION_ACCEPT_INPUT delayed via the application queue. This functionality is only reflected in the current application; if the cursor roams outside the application bounds, the default behavior will occur.

GA_READ_ONLY

This flag indicates that this object's only function is to display information to the user; the user will not be able to interact with this object. This attribute is set most often in lists or text objects.

GA_KBD_SEARCH_PATH

This flag indicates that this generic branch contains objects with keyboard accelerators and should therefore be searched when evaluating such events. This attribute bit is set internally by the system. There should be no need for your application to deal with this attribute directly.

GA_TARGETABLE

This flag indicates that this object is targetable and is eligible to receive the target exclusive within its target level. Most specific UIs will automatically grab the target for this object whenever the user interacts with it. This attribute is set by default within the following classes:

GenField
GenApplication
GenPrimary
GenDisplayControl
GenDisplay
GenView

GA_NOTIFY_VISIBILITY
This flag indicates that this object should send notification when it becomes visible and not visible. Objects thus will send MSG_GEN_APPLICATION_VISIBILITY_NOTIFICATION to the GenApplication object whenever the state of their visibility changes (see “GenApplication,” Chapter 3). You may alter this behavior by including one of the visibility vardata attributes.

Code Display 2-15 Using GI_attrs in a Dialog Box

```
@object GenInteractionClass MyDialogBox = {
    GI_comp = @MyButton, @MyOtherButton;
    GII_visibility = GIV_DIALOG; /* build this Interaction as a dialog box.*/
}

@object GenTriggerClass MyButton = {
    GTI_actionMsg = MSG_MY_SPECIAL_MESSAGE;
    GTI_destination = process;
        /* MyButton, when activated, will send the message above to the
         * process object. Only when that happens will it activate the
         * behavior within the GI_attrs instance data below. */
    GI_attrs = @default |
        /* This flag will close the MyDialogBox object */
        GA_SIGNAL_INTERACTION_COMPLETE |
        /* This flag will set the application to ignore all input events while the
         * message above is processed. */
        GA_INITIATES_INPUT_IGNORE;
}
```

■ MSG_GEN_GET_ATTRIBUTES

```
byte    MSG_GEN_GET_ATTRIBUTES();
```

This message retrieves the *GI_attrs* instance data for the object the message is sent to. This message returns a byte length bitfield.

Source: Unrestricted.

Destination: Any generic object.

Return: Byte length *GI_attrs* bitfield.

Interception: Generally not intercepted.

■ MSG_GEN_SET_ATTRS

```
void        MSG_GEN_SET_ATTRS(
            byte attrsToSet,
            byte attrsToClear);
```

This message sets the recipient's *GI_attrs* field. This message takes two arguments: the attributes to set and the attributes to clear. There is no need to repeat attributes that have been previously set. Note that these attributes will not take effect until the object is activated in the normal manner. (Sending this message does not in itself initiate the activity described).

2.3

You should only send this message while an object is not GS_USABLE, because these attributes are only checked when an object is specifically built. Setting the attributes of a GS_USABLE object may cause an error.

Source: Unrestricted.

Destination: Any non-usable generic object.

Parameters: *attrsToSet* **GenAttributes** to set in *GI_attrs*.

attrsToClear **GenAttributes** to clear in *GI_attrs*.

Interception: Generally not intercepted.

Code Display 2-16 Conditionally Altering the GI_attrs Field

```
@method MyProcessClass, MSG_DO_CUSTOM_ATTRS {
    byte MyAttrs;

        /* retrieve the GI_attrs field */
    MyAttrs = @call @MyObject::MSG_GEN_GET_ATTRIBUTES();

        /* If the GA_COMPLETES_INTERACTION bit is set, then set it
         * GA_INITIATES_BUSY_STATE also. Otherwise set it
         * GA_INITIATES_INPUT_IGNORE. */

    if (MyAttrs & GA_COMPLETES_INTERACTION){
        @call @MyObject::MSG_GEN_SET_NOT_USABLE();
        @call @MyObject::MSG_GEN_SET_ATTRS(GA_INITIATES_BUSY_STATE, 0);

        /* Note that setting an object's GA_INITIATES_BUSY_STATE attribute will
         * not in itself initiate a busy state. That object will only issue a busy
         * state when it is activated in the normal fashion. */
    @call @MyObject::MSG_GEN_SET_USABLE();
    } else {
        @call @MyObject::MSG_GEN_SET_NOT_USABLE();
        @call @MyObject::MSG_GEN_SET_ATTRS(GA_INITITATES_INPUT_IGNORE, 0);
        @call @MyObject::MSG_GEN_SET_USABLE();
    }
}
```

2.3.5 States

`GI_states`

The *GI_states* attribute is a record that holds the state of the generic object. These states affect the object's visual representation and its functionality. By default, all objects have both GS_USABLE and GS_ENABLED set when first built. There are two *GI_states*:

GS_USABLE This state controls the usability of an object. Setting an object GS_USABLE will indicate that both the object and the entire generic branch below this object should be considered as an active part of the user interface and therefore should be visually represented. An object that is not set GS_USABLE cannot appear as part of the user interface. Before an object can be visually built and considered part of the user interface, however, it must satisfy two conditions: The object itself must be GS_USABLE, and all ancestors (parents) of the object must also be GS_USABLE.

This means that a continuous path of GS_USABLE objects must exist from the top node of your application to the object for it to be usable. In this case, an object is considered to be *fully* usable. Conversely, if an object is not GS_USABLE, then no object in the branch below it can be visually displayed either (even if it is GS_USABLE). Therefore, an object may be set GS_USABLE without being fully usable. This allows you to set an entire branch fully usable by setting just one node object usable, provided that all other objects below that node had previously been set GS_USABLE. This attribute is set by default. (See Figure 2-9 for an example.)

Destination Menu
Fixed Child
Fixed Child
Destination Menu
Fixed Child
Usable Child
Fixed Child

Figure 2-9 *Setting an Object GS_USABLE*

When the GenTrigger object "Usable Child" is set GS_USABLE, it is built into the menu.

GS_ENABLED

This state sets the object enabled. This state controls whether the object is available to the user for activity. An object must be enabled for the user to interact with it. If an object is set GS_USABLE but not GS_ENABLED, it will be visually displayed but will not accept input from the user. The object will also be unable to perform its normal activities. This is usually represented by "graying out" the object's visual moniker. Before an object can be "enabled" for user input and activation, however, it must satisfy two conditions: The object must be GS_ENABLED; all ancestors (parents) of the object must also be GS_ENABLED. Therefore, a continuous path of GS_ENABLED objects must exist from the top node of your application to the object in question for it to be enabled. In this case, an object is considered to be *fully* enabled.

Conversely, if an object is not GS_ENABLED, then no object in the branch below it can be interacted with either (even if it is GS_ENABLED). Therefore, an object may be GS_ENABLED without being fully enabled. This allows you to set an entire branch fully enabled by setting just one node object enabled, provided that all other objects below that node had previously been set GS_ENABLED. See Figure 2-10 for an example.

These states not only affect the state of the current object but all of its descendants as well. When your user interface is first built (or rebuilt) it conducts a top-down search, building any objects that are declared GS_USABLE. If an object is not GS_USABLE, it will not be visually built. Furthermore, none of its descendants will be searched (or built). Therefore, if any single node is not GS_USABLE, no other object below that node will be fully usable.

In cases where you wish to alter any fundamental behavior of an object, you may have to set the object not usable, change its behavior, and then set the object GS_USABLE again. This ensures that the object is built out correctly (including visually) according to the new criteria.

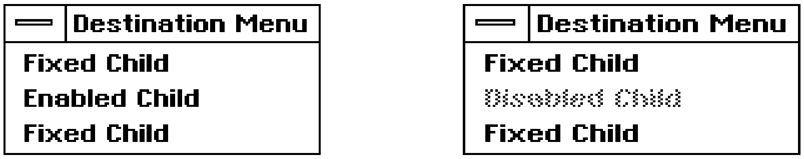


Figure 2-10 *Setting an Object GS_ENABLED*
When the GenTrigger object "Enabled Child" is set not GS_ENABLED, it is grayed out (disabled).

Code Display 2-17 Setting GI_states

```
@object GenTriggerClass MyTrigger = {
                /* The default GI_states are GS_USABLE and GS_ENABLED.
                 * This object will only be GS_USABLE */
    GI_states = @default & ~GS_ENABLED;
}
```

2.3.5.1 The Usable State

MSG_GEN_GET_USABLE, MSG_GEN_SET_USABLE, MSG_GEN_SET_NOT_USABLE, MSG_GEN_CHECK_IF_FULLY_USABLE

Setting an object GS_USABLE will incorporate the object into the generic tree and regard the object as part of the user interface. (It will not by itself enable an object for user input; this requires an object to be set GS_ENABLED also.) If an object is not GS_USABLE, it cannot be used in any manner including any visual implementation by the user interface. The following messages manipulate an object's usable state.

■ MSG_GEN_GET_USABLE

```
Boolean   MSG_GEN_GET_USABLE();
```

This message checks the GS_USABLE bit of the *GI_states* field.

Source: Unrestricted.

Destination: Any generic object.

Parameters: None.

Return: Will return *true* if the object is GS_USABLE, *false* if it is not usable.

Interception: Generally not intercepted.

■ MSG_GEN_SET_USABLE

```
void      MSG_GEN_SET_USABLE(
          VisUpdateMode updateMode);
```

This message sets an object GS_USABLE. (This message has no effect on an object already GS_USABLE.) Objects may be set GS_USABLE only after they have been attached to a generic tree. Setting an object usable forces that object to be specifically built. If the object's associated window is realized, the object will be visually built and updated. Setting an object GS_USABLE that is not attached to the generic tree will cause an error.

Setting an object usable allows that object to become part of the user interface. Before the object can be used, though, it must be *fully usable*. An object becomes fully usable only if all of its ancestors are GS_USABLE. If any ancestor is not GS_USABLE, then the entire branch below it will not be fully usable. You can check if an object is fully usable with the message MSG_GEN_CHECK_IF_FULLY_USABLE.

Conversely, setting a node GS_USABLE, in which every branch object below it is already GS_USABLE will make that entire branch fully usable. This is useful for bringing up an entire section of a generic tree by setting a single object usable.

Source: Unrestricted.

Destination: Any generic object.

Parameters: *updateMode* **VisUpdateMode** to use when updating changes to the screen.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_SET_NOT_USABLE

```
void    MSG_GEN_SET_NOT_USABLE(
        VisUpdateMode updateMode);
```

This message sets an object not usable (clears the GS_USABLE bit). Because an object may be visually unbuilt by this message, the **VisUpdateMode** VUM_MANUAL is not allowed.

Source: Unrestricted.

Destination: Any generic object.

Parameters: *updateMode* **VisUpdateMode** to use when updating changes to the screen. May not be VUM_MANUAL.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_CHECK_IF_FULLY_USABLE

```
Boolean MSG_GEN_CHECK_IF_FULLY_USABLE();
```

This message checks whether an object is fully usable. The object and all of its parents must be GS_USABLE for the object to be fully usable.

Source: Unrestricted.

Destination: Any generic object.

Return: Will return *true* if the object is fully usable, *false* if it is not.

Interception: Generally not intercepted.

2.3.5.2 The Enabled State

MSG_GEN_GET_ENABLED, MSG_GEN_SET_ENABLED, MSG_GEN_SET_NOT_ENABLED, MSG_GEN_CHECK_IF_FULLY_ENABLED

An object that is GS_ENABLED is ready for user interaction. As in the case with GS_USABLE, all parents of the object in question must be GS_ENABLED to *fully enable* the object. An object may be fully enabled without being fully usable, but the object will not be visually represented. You can check if an object is fully enabled with MSG_GEN_CHECK_IF_FULLY_ENABLED.

An object that is not enabled but is set usable will be represented in the user interface; it will not, however, allow the user to interact with it. In many cases, the specific UI implements this by “graying out” the object. Setting a disabled object will visually update it.

■ MSG_GEN_GET_ENABLED

```
Boolean   MSG_GEN_GET_ENABLED();
```

This message returns the enabled state of the object the message is sent to.

Source: Unrestricted.

Destination: Any generic object.

Return: Will return *true* if the object is enabled, *false* if it is not.

Interception: Generally not intercepted.

■ MSG_GEN_SET_ENABLED

```
void      MSG_GEN_SET_ENABLED(
          VisUpdateMode updateMode);
```

This message sets an object GS_ENABLED. (This message has no effect on an object already GS_ENABLED.) You must pass this message a **VisUpdateMode**. Sending this message allows the object to receive user input. Only send this message if your application will be ready to interact with the object after the period specified in the **VisUpdateMode** has passed.

Source: Unrestricted.

Destination: Any generic object.

Parameters: *updateMode* **VisUpdateMode** to use when updating changes to the screen.

Interception: Generally not intercepted.

■ MSG_GEN_SET_NOT_ENABLED

```
void      MSG_GEN_SET_NOT_ENABLED(
          VisUpdateMode updateMode);
```

This message sets the object not enabled (clears the object’s GS_ENABLED bit.) You must pass the message a **VisUpdateMode**. In most specific UIs a disabled state is implemented by “graying out” the object. The user will be unable to interact with the object according to the **VisUpdateMode** passed.

Source: Unrestricted.

Destination: Any generic object.

Parameters: *updateMode* — **VisUpdateMode** to use when updating changes to the screen. May not be VUM_MANUAL.

Interception: Generally not intercepted.

■ MSG_GEN_CHECK_IF_FULLY_ENABLED

```
Boolean   MSG_GEN_CHECK_IF_FULLY_ENABLED();
```

This message checks whether an object is fully enabled. An object is only fully enabled (ready for user interaction) when both it and all of its ancestors are enabled.

Source: Unrestricted.

Destination: Any generic object.

Return: Will return *true* if fully enabled, *false* if not.

Interception: Generally not intercepted.

2.4 Modifying GenClass Instance Data

Instance data need not be set solely within your **.goc** file. You may also retrieve or alter this data dynamically with messages in **GenClass**. The following section details messages to modify the *GI_visMoniker*, *GI_kbdAccelerator*, *GI_attrs* and *GI_states* instance fields. This section also details ways of using the variable data object routines and messages to modify a generic object's vardata.

Remember that when accessing and modifying instance data, you may need to dereference the pointer to the object's instance data if the location of that instance data may have changed. This pointer (**pself**) is automatically set up correctly inside a method upon receipt of the corresponding message. If you perform actions within that method that may move the object, you can use **ObjDerefGen**() to dereference **pself** again.

If the instance fields you wish to modify are related to generic tree mechanisms (*GI_comp* and *GI_link*), see “Special Message Passing” on page 251.

2.4.1 Visual Monikers

A visual moniker must reside in a local memory (LMem) chunk. That chunk should be located within the object block of the particular generic object it is associated with. Once monikers are defined, there are several messages in **GenClass** that you can use to change the visual moniker for an object.

The *GI_visMoniker* field contains a ChunkHandle of the **VisMoniker** structure. The **VisMoniker** structure contains three pieces of data: the type of moniker, the width of the moniker chunk, and the moniker data.

2.4.2 Managing Visual Monikers

```
MSG_GEN_GET_VIS_MONIKER, MSG_GEN_USE_VIS_MONIKER,
MSG_GEN_CREATE_VIS_MONIKER, MSG_GEN_REPLACE_VIS_MONIKER,
MSG_GEN_REPLACE_VIS_MONIKER_OPTR,
MSG_GEN_REPLACE_VIS_MONIKER_TEXT, MSG_GEN_FIND_MONIKER,
MSG_GEN_DRAW_MONIKER, MSG_GEN_GET_MONIKER_POS,
MSG_GEN_GET_MONIKER_SIZE, MSG_GEN_RELOC_MONIKER_LIST,
MSG_GEN_FIND_OBJECT_WITH_TEXT_MONIKER
```

Sending MSG_GEN_GET_VIS_MONIKER to an object returns the chunk handle of the visual moniker (*GI_visMoniker*) for that object. You can then inspect that visual moniker or use it within other objects (though this is not recommended). You must be careful when sharing monikers with other objects, as freeing one object (and therefore its associated moniker) will interfere with any other objects referencing that moniker.

Sending MSG_GEN_USE_VIS_MONIKER to an object sets a visual moniker for that object to use. The message must pass the chunk handle of the moniker desired for the object along with a **VisUpdateMode**. Valid **VisUpdateMode**s are described in "VisClass," Chapter 23. The moniker to use must reside in the same block as the object being set. If you need to set an object to use a new moniker from a location outside of its object block, use MSG_GEN_REPLACE_VIS_MONIKER to copy the moniker to the new location.

MSG_GEN_REPLACE_VIS_MONIKER copies a visual moniker from a source location into a destination. This message must pass parameters specifying the type of copy operation. The source for the moniker can be referenced using an optr or a simple text string. Use this message when you wish to copy a moniker from outside of the object's current object block.

MSG_GEN_REPLACE_VIS_MONIKER_OPTR and MSG_GEN_REPLACE_VIS_MONIKER_TEXT are simplified versions of the above message, allowing you to set a visual moniker from an optr or from a pointer to a null-terminated text string without having to pass many other arguments.

MSG_GEN_CREATE_VIS_MONIKER creates a visual moniker (within the resource block of the object this message is sent to), copying it from a source; it does not, however, attach this moniker to any object.

MSG_GEN_FIND_MONIKER searches for a specific type of moniker within a moniker list. You may then use the moniker directly. You can also replace the original moniker list with the specified moniker.

MSG_GEN_DRAW_MONIKER, MSG_GEN_GET_MONIKER_POS, and MSG_GEN_GET_MONIKER_SIZE all manipulate a currently existing moniker. These messages are most useful for custom gadgets. MSG_GEN_DRAW_MONIKER draws the moniker according to the criteria passed with the message. MSG_GEN_GET_MONIKER_POS and MSG_GEN_GET_MONIKER_SIZE return GEOS coordinates specifying the location and size of the moniker, respectively. You may then inspect and alter these coordinates before sending the moniker a MSG_GEN_DRAW_MONIKER to redraw the moniker according to the changed criteria.

The utility message MSG_GEN_RELOC_MONIKER_LIST relocates an object's moniker list.

To find an object with a particular visual moniker, send MSG_GEN_FIND_OBJECT_WITH_MONIKER_FLAGS to the object at which to start the top-down search, passing the text to perform the match.

■ MSG_GEN_GET_VIS_MONIKER

```
ChunkHandle MSG_GEN_GET_VIS_MONIKER();
```

This message retrieves the instance data in the object's current *GI_visMoniker* instance field. This message returns the ChunkHandle of the moniker data structure. You can then use that chunk handle to manipulate the visual moniker directly or to copy the moniker for use by other objects.

Source: Unrestricted.

2.4

Destination: Any generic object.

Return: The chunk handle of the visual moniker in use by this object.

Interception: Generally not intercepted.

■ MSG_GEN_USE_VIS_MONIKER

```
void        MSG_GEN_USE_VIS_MONIKER(
            ChunkHandle moniker,
            VisUpdateMode updateMode);
```

This message allows an object to reference a visual moniker; this message does not destroy the chunk of the object's current visual moniker. The moniker must reside in the same block as the object. Use MSG_GEN_REPLACE_VIS_MONIKER if you wish to set a moniker from a source outside the object block or if you wish to use a new moniker, overwriting the old one.

You cannot pass a moniker list's chunk handle using this message. See MSG_GEN_FIND_MONIKER for information on selecting monikers from within a list.

Source: Unrestricted.

Destination: Any generic object.

Parameters:

moniker	The chunk handle of the visual moniker to use
updateMode	**VisUpdateMode** to determine when the moniker will be redrawn.

Return: Nothing.

Interception: Generally not intercepted.

◆ Objects

Code Display 2-18 Getting and Using Visual Monikers

```
/* This method effectively copies a visual moniker from ObjectOne into ObjectTwo.
 * Both objects must reside in the same object block. Note that in effect, both
 * ObjectOne and ObjectTwo will “share” the same moniker. This can be dangerous if
 * one object is freed, thereby causing the other object to become dereferenced.
 * For more complex copy operations, use MSG_GEN_REPLACE_VIS_MONIKER instead. */

@method MyProcessClass, MSG_MY_MONIKER_MESSAGE {
        /* Set up variable to store the chunk handle of the visual moniker. */
    ChunkHandle MyVisMonikerCH;

        /* Retrieve the visMoniker of ObjectOne and store it in the variable. */

    MyVisMonikerCH = @call @ObjectOne::MSG_GEN_GET_VIS_MONIKER();

        /* Set the visMoniker of Object Two to MyVisMonikerCH and update
         * immediately. Use @send if you don’t expect a return value. */

    @send @ObjectTwo::MSG_GEN_USE_VIS_MONIKER(MyVisMonikerCH, VUM_NOW);
}

/* You can also declare visual monikers in your .goc file and set them later */

    @visMoniker OnMoniker = “On”;
    @visMoniker OffMoniker = “Off”;

        /* Within a method, the following routine will set the moniker to “On”. The
         * OptrToChunk operator casts the object from an optr to a ChunkHandle. */

    @call @MyObject::MSG_GEN_SET_MONIKER(OptrToChunk(@OnMoniker), VUM_NOW);

        /* Within a method, the following routine will set the moniker to “Off” */

    @call @MyObject::MSG_GEN_SET_MONIKER(OptrToChunk(@OffMoniker), VUM_NOW);
```

■ MSG_GEN_REPLACE_VIS_MONIKER

```
ChunkHandle MSG_GEN_REPLACE_VIS_MONIKER(@stack
          VisUpdateMode          updateMode,
          word                   height,
          word                   width,
          word                   length,
          VisMonikerDataType     dataType,
          VisMonikerSourceType   sourceType,
          dword                  source);
```

This message copies a visual moniker from a source to the object sent the message. The message returns the chunk handle of the newly copied moniker. The object's current visual moniker is overwritten by this message; therefore, if more than one object shares the overwritten block, the other object should be updated with MSG_GEN_USE_VIS_MONIKER. This message can copy a visual moniker from a variety of sources.

This message is a general, all-purpose, moniker replacement device. If your visual moniker source is a **visMoniker** structure or a null-terminated text string, you may be able to use the simpler messages MSG_GEN_REPLACE_VIS_MONIKER_OPTR and MSG_GEN_REPLACE_VIS_MONIKER_TEXT.

Source: Unrestricted.

Destination: Any generic object.

Parameters: *updateMode* Specifies the **VisUpdateMode** to use when visually updating the changes to the screen. See "VisClass," Chapter 23 for more information on **VisUpdateMode** types.

height If moniker is a GString, this specifies the height of the moniker in points. Otherwise, set this to zero.

width If the moniker is a GString, this specifies the width of the moniker in points. Otherwise, set this to zero.

length If the moniker is a GString, this specifies the length of the moniker in points. Otherwise, set this to zero.

dataType Specifies the type of data referenced by the *source*. Valid **VisMonikerDataType** types include VMDT_VIS_MONIKER, VMDT_TEXT, VMDT_GSTRING, or VMDT_TOKEN.

sourceType Specifies the type of pointer referencing the *source*. Valid **VisMonikerSourceType** types are VMST_FPTR, VMST_OPTR, and VMST_HPTR.

source Specifies the source of the moniker to be used during the copy process. This source may be a pointer to a text string, an optr pointing to a visual moniker outside the current block, or a variety of other sources. The source type should be specified in *sourceType*.

Return: The chunk handle of the new visual moniker for the object.

Interception: Generally not intercepted.

■ MSG_GEN_REPLACE_VIS_MONIKER_OPTR

```
ChunkHandle MSG_GEN_REPLACE_VIS_MONIKER_OPTR(
          optr                source,
          VisUpdateMode       updateMode);
```

This message is a simplified version of MSG_GEN_REPLACE_VIS_MONIKER. If your visual moniker source is an existing **visMoniker** structure, you can use this message to replace an object's moniker.

Source: Unrestricted.

Destination: Any generic object.

Parameters: *source* The optr of the visMoniker structure to use in the replacement operation.

updateMode Specifies the **VisUpdateMode** to use when visually updating the changes to the screen.

Return: The chunk handle of the new visual moniker for the object.

Interception: Generally not intercepted.

■ MSG_GEN_REPLACE_VIS_MONIKER_TEXT

```
ChunkHandle MSG_GEN_REPLACE_VIS_MONIKER_TEXT(
          const char          *source,
          VisUpdateMode       updateMode);
```

This message is a simplified version of MSG_GEN_REPLACE_VIS_MONIKER. If your visual moniker source is a pointer to a null-terminated text string, you can use this message to replace an object's moniker without having to pass the arguments of MSG_GEN_REPLACE_VIS_MONIKER.

Source: Unrestricted.

Destination: Any generic object.

Parameters: *source* A pointer to a null-terminated character string to use in the replacement operation.

updateMode Specifies the **VisUpdateMode** to use when visually updating the changes to the screen.

Return: The chunk handle of the new visual moniker for the object.

Interception: Generally not intercepted.

■ MSG_GEN_CREATE_VIS_MONIKER

```
ChunkHandle MSG_GEN_CREATE_VIS_MONIKER(@stack
          CreateVisMonikerFlags  flags,
          word                   height,
          word                   width,
          word                   length,
          VisMonikerDataType     dataType,
          VisMonikerSourceType   sourceType,
          dword                  source);
```

This message creates a visual moniker chunk within the resource block of the object sent this message. The moniker is copied from the source specified but is not attached to any object in the new object block.

Source: Unrestricted

Destination: Any **GenClass** object

Parameters: *flags* Flags to use in the creation of the new visual moniker. CVMF_DIRTY marks the new moniker chunk OCF_DIRTY.

height If moniker is a GString, this specifies the height of the moniker in points. Otherwise, set this to zero.

width If the moniker is a GString, this specifies the width of the moniker in points. Otherwise, set this to zero.

length If the moniker is a GString, this specifies the length of the moniker in points. Otherwise, set this to zero.

dataType Specifies the type of data referenced by the *source*. Valid **VisMonikerDataType** types include VMDT_VIS_MONIKER, VMDT_TEXT, VMDT_GSTRING, or VMDT_TOKEN.

sourceType Specifies the type of pointer referencing the *source*. Valid **VisMonikerSourceType** types are VMST_FPTR, VMST_OPTR, and VMST_HPTR.

source Specifies the source of the moniker to be used during the copy process. This source may be a pointer to a text string, an optr pointing to a visual moniker outside the current block, or a variety of other sources. The source type should be specified in *sourceType*.

Return: The chunk handle of the new visual moniker for the object.

Interception: Generally not intercepted.

■ MSG_GEN_FIND_MONIKER

```
optr      MSG_GEN_FIND_MONIKER(
          Boolean             useAppMonikerList,
          word                searchFlags,     /* VisMonikerSearchFlags */
          MemHandle           destBlock);
```

This message scans a list of monikers and selects a specific moniker (or the most appropriate moniker). The moniker selected is determined according to the passed **VisMonikerSearchFlags**. This message is usually used by the specific UI to select an application moniker from a list.

If the first argument (*useAppMonikerList*) is not zero, the message will use the GenApplication's moniker list instead of the recipient's. This is useful for finding the icon used for an iconified application.

Note that this message returns an optr, not a chunk handle, as this message is most often used to find a moniker outside of the current object block. This message returns zero if no moniker is found.

Source: Unrestricted.

Destination: Any generic object.

Parameters: *useAppMonikerList*

Non-zero to use the moniker list of the application associated with this object.

searchFlags **VisMonikerSearchFlags** to use for the search criteria. If VMSF_COPY_CHUNK is specified in the *searchFlags*, a block handle must be supplied in *destBlock*. The selected moniker will be copied into that block.

destBlock Block to copy a moniker into, if VMSF_COPY_CHUNK is specified in the *searchFlags*.

Return: The optr of the moniker found according to the search criteria. (This may be outside the current object block.)

Interception: Generally not intercepted.

■ MSG_GEN_DRAW_MONIKER

```
void        MSG_GEN_DRAW_MONIKER(@stack
            DrawMonikerFlags    monikerFlags,
            word                textHeight,
            GStateHandle        gState,
            word                yMaximum,
            word                xMaximum,
            word                yInset,
            word                xInset);
```

This message draws a moniker for an object according to the parameters passed. The moniker must currently exist. You can use this message to change the way you wish a visual moniker to appear within a generic object without actually changing the visual moniker itself.

Source: Unrestricted.

Destination: Any GS_USABLE generic object.

Parameters: *monikerFlags* Specifies the **DrawMonikerFlags** to use when drawing the moniker. The **DrawMonikerFlags** record specifies the conditions to draw the moniker under.

textHeight Specifies the height of the system text. Passing this value speeds the processing of this message. If you do not know the height of the system text, pass zero.

gState Specifies the GState, if any, to use when drawing the moniker.

yMaximum, xMaximum
These specify the maximum height and width of the moniker. Typically, these are used with

DMF_CLIP_TO_MAX_WIDTH (in **DrawMonikerFlags**) to perform a clipping operation on the moniker. Pass zero if no maximum width is desired.

yInset Specifies the point to begin drawing the moniker if the moniker is right or left justified.

xInset Specifies the point to begin drawing the moniker if the moniker is top or bottom justified.

Return: Nothing

Interception: Generally not intercepted. Custom gadgets may handle this if they are supplementing or replacing default functionality.

Structures: The **DrawMonikerFlags** record defines the parameters to use when drawing a currently existing visual moniker. This record is used by MSG_GEN_DRAW_MONIKER, MSG_GEN_GET_MONIKER_POS and MSG_GEN_GET_MONIKER_SIZE. Its flags are shown below:

```
DMF_UNDERLINE_ACCELERATOR
DMF_CLIP_TO_MAX_WIDTH
DMF_NONE
DMF_Y_JUST_MASK    /* of type Justifications */
DMF_X_JUST_MASK    /* of type Justifications */
```

Use DMF_UNDERLINE_ACCELERATOR to underline the keyboard accelerator of a text moniker. Use DMF_CLIP_TO_MAX_WIDTH to clip the visual bounds of the moniker to the width specified in *maxWidth***.** Use DMF_NONE to draw the moniker at the current pen position.

The two justification masks are of type **Justification**, and they specify the vertical and horizontal justifications to use when drawing the moniker. The values you can use in the justification fields are

```
J_LEFT
J_RIGHT
J_CENTER
J_FULL
```

■ MSG_GEN_GET_MONIKER_POS

```
XYValueASDWord MSG_GEN_GET_MONIKER_POS(@stack
            DrawMonikerFlags    monikerFlags,
            word                textHeight,
            GStateHandle        gState,
            word                yMaximum,
            word                xMaximum,
            word                yInset,
            word                xInset);
```

2.4

This message returns the x and y coordinates of the moniker for the object as if the moniker were drawn according to the passed parameters (the same as for MSG_GEN_DRAW_MONIKER). Use the macros DWORD_X and DWORD_Y to extract the appropriate coordinates from the returned value.

This message, along with MSG_GEN_DRAW_MONIKER and MSG_GEN_GET_MONIKER_SIZE is most useful in the construction and modification of custom gadgets.

Source: Unrestricted.

Destination: Any generic object.

Parameters: See MSG_GEN_DRAW_MONIKER.

Return: **XYValueAsDWord** which can be split up into x and y coordinates of the moniker position with the macros DWORD_X and DWORD_Y.

Interception: Generally not intercepted.

■ MSG_GEN_GET_MONIKER_SIZE

```
SizeAsDWord MSG_GEN_GET_MONIKER_SIZE(
            word                textHeight,
            GStateHandle        gState);
```

This message returns the width and height (in points) of the moniker for the object. Use the macros DWORD_WIDTH and DWORD_HEIGHT to extract the appropriate coordinates from the returned value. This message, along with MSG_GEN_DRAW_MONIKER and MSG_GEN_GET_MONIKER_POS, is most useful in the construction and modification of custom gadgets.

Source: Unrestricted.

Destination: Any generic object.

Parameters: *textHeight* The system text height, if known; otherwise, pass zero.

gstate The GState handle of the GState to draw to.

Return: **SizeAsDWord** which can be split up into with and height with the macros DWORD_WIDTH and DWORD_HEIGHT.

Interception: Generally not intercepted.

■ MSG_GEN_RELOC_MONIKER_LIST

```
void      MSG_GEN_RELOC_MONIKER_LIST(
          optr               monikerList,
          word               relocFlag); /* 0: relocate; 1: unrelocate */
```

This method (un)relocates a passed moniker list.

Source: Anyone.

Destination: Any GenClass object.

Parameters: *monikerList* optr of moniker list.

reloc Flag specifying whether relocation or unrelocation is desired.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_FIND_OBJECT_WITH_TEXT_MONIKER

```
optr      MSG_GEN_FIND_OBJECT_WITH_TEXT_MONIKER(
          char   *text,
          word   flags);
```

This message recursively searches through the generic tree (starting at the object sent the message) for an object whose visual moniker matches the passed text. This match does not need to be complete (unless the flag GFTMF_EXACT_MATCH is passed); the message will return the first object whose initial characters match the given text. The message must pass **GenFindObjectWithMonikerFlags**.

```
typedef WordFlags GenFindObjectWithMonikerFlags;
#define GFTMF_EXACT_MATCH 0x8000
#define GFTMF_SKIP_THIS_NODE 0x4000
```

If GFTMF_EXACT_MATCH is passed, the text and the visual moniker text must match exactly (null-terminated). If GFTMF_SKIP_THIS_NODE is passed, the object sent the message will not be checked; only it's children will be checked.

Source: Unrestricted.

Destination: Any generic object.

Parameters: *text* Null-terminated text.

flags **GenFindObjectWithMonikerFlags**.

Return: optr of the object with the matching moniker, or null if no moniker was found.

Interception: Generally not intercepted.

2.4

2.4.3 Managing Keyboard Accelerators

`MSG_GEN_GET_KBD_ACCELERATOR, MSG_GEN_SET_KBD_ACCELERATOR, MSG_GEN_CHANGE_ACCELERATOR`

Keyboard Accelerators provide keyboard shortcuts for heavily used application functions. For complete information on valid *GI_kbdAccelerator* instance data and the functionality those data provide, see “Keyboard Accelerators” on page 184.

MSG_GEN_GET_KBD_ACCELERATOR returns a word of data specifying the current keyboard accelerator, if any, for the object the message is sent to.

MSG_GEN_SET_KBD_ACCELERATOR sets the keyboard accelerator of the object the message is sent to. The message must pass the keyboard accelerator to use along with a **VisUpdateMode**. If you wish to remove a keyboard accelerator from an object, pass this message with a keyboard acclerator of zero.

MSG_GEN_CHANGE_ACCELERATOR changes the keyboard accelerator using a low-level replace operation.

■ MSG_GEN_GET_KBD_ACCLERATOR

```
word      MSG_GEN_GET_KBD_ACCELERATOR();
```

This message returns the object’s current *GI_kbdAccelerator*. The message returns a word-length data structure (**KeyboardShortcut**). You can then manipulate or use this word of data to set other object’s keyboard accelerators. Make sure that you do not have two objects sharing the same keyboard accelerator.

Source: Unrestricted.

Destination: Any generic object.

Return: **KeyboardShortcut** of the generic object (in *GI_kbdAccelerator*).

Interception: Generally not intercepted.

■ MSG_GEN_SET_KBD_ACCELERATOR

```
void    MSG_GEN_SET_KBD_ACCELERATOR(
        word                accelerator,
        VisUpdateMode       updateMode);
```

This message sets the *GI_kbdAccelerator* of the object the message is sent to.

Source: Unrestricted.

Destination: Any generic object.

Parameters: *accelerator* — **KeyboardShortcut** to set *GI_kbdAccelerator* of the generic object to.

updateMode — **VisUpdateMode** to determine when redrawing occurs.

Interception: Generally not intercepted.

Code Display 2-19 Using a Keyboard Accelerator

```
@kbdAccelerator ShortcutOne = ctrl shift ‘k’;

        /* @specificUI keyword needed to use DELETE key */
@kbdAccelerator ShortcutTwo = @specificUI ctrl DELETE;

@object GenTriggerClass MyTrigger = {
    GI_kbdAccelerator = ShortcutOne;/* sets accel to “ctrl shift ‘k’” */
}

@object GenTriggerClass MyOtherTrigger = {
        GI_kbdAccelerator = ShortcutCopy;/* Copies “ctrl shift ‘k’” to
                                          * MyOtherTrigger */
}
```

■ MSG_GEN_CHANGE_ACCELERATOR

```
void      MSG_GEN_CHANGE_ACCELERATOR(
          word   bitsToClear,
          word   bitsToSet);
```

This message changes the keyboard accelerator using constants defined in the input library. In most cases, use MSG_GEN_SET_KBD_ACCELERATOR instead. MSG_GEN_CHANGE_ACCELERATOR is useful in some cases where you wish to tinker slightly with the keyboard shortcut's usage (as in a case where an accelerator of ctrl z should change to alt z in the duplication of a template object).

Source: Unrestricted.

Destination: Any generic object.

Parameters: *bitsToClear* **KeyboardShortcut** and/or **VChar** to clear.

bitsToSet **KeyboardShortcut** and/or **VChar** to set.

Interception: Generally not intercepted.

2.5 Generic Trees

Generic trees are constructed in several ways. The most common is to add, remove, and move individual children. These children may be the top of a generic branch and will therefore add, remove, or move the entire branch below them. **GenClass** also provides other means of manipulating entire branches of generic trees. Each manner of generic tree construction and destruction has certain advantages and disadvantages. The various methods are detailed below:

- The object may be declared in the **.goc** file, attached to the generic tree. In this case, there is no need to add the child as this is taken care of by the compiler. You can remove the object and any of its children with MSG_GEN_REMOVE (or MSG_GEN_REMOVE_CHILD). After removal, any single object may be destroyed with MSG_META_OBJ_FREE. You may also destroy a tree of generic objects with MSG_GEN_DESTROY. You may add the removed object (and its children) to a different location in the generic tree with MSG_GEN_ADD_CHILD.

- The object may exist in your **.goc** file but remain unattached to the generic tree. You may attach the object with MSG_GEN_ADD_CHILD. You can remove the object, and any children of this object, with MSG_GEN_REMOVE (or MSG_GEN_REMOVE_CHILD). After removal, destroy any single object with MSG_META_OBJ_FREE. You can destroy a tree of generic objects with MSG_GEN_DESTROY.

- **ObjInstantiate()**
  This routine creates a new instance of a single object. You can then add the object to your generic tree with MSG_GEN_ADD_CHILD. After a child is removed using MSG_GEN_REMOVE (or MSG_GEN_REMOVE_CHILD), it can be destroyed using MSG_META_OBJ_FREE. The object destroyed in such a manner must not have any children attached to it at the time of destruction.

- **ObjDuplicateResource()**
  This routine duplicates an instance of a previously defined object resource. That object block may contain one or more objects. Only use this routine if you want to duplicate an instance of a predefined resource block. After a block duplicated in this manner is removed using MSG_GEN_REMOVE (or MSG_GEN_REMOVE_CHILD), it can be destroyed using MSG_FREE_DUPLICATE.

- MSG_GEN_COPY_TREE
  This message copies an entire generic branch headed by the recipient. The branch will be set usable and added automatically without need for a MSG_GEN_ADD_CHILD. A generic branch created in this manner can be removed with MSG_GEN_DESTROY. These messages are useful because they can duplicate and destroy generic trees across object blocks. (The copied branch will reside in one block, however.)

Objects created in these ways must be added using MSG_GEN_ADD_CHILD (except for those created with MSG_GEN_COPY_TREE).

- MSG_GEN_ADD_CHILD
  This message adds a generic object that was previously created but not attached to the generic tree. The object may have been created in your **.goc** file but not attached as a child of any other object. The object may also have been created through **ObjInstantiate()** or with **ObjDuplicateResource()** (see above). Use MSG_GEN_REMOVE (or MSG_GEN_REMOVE_CHILD) to remove a child added in this manner.

2.5.1 Child/Parent Searches

MSG_GEN_FIND_CHILD, MSG_GEN_FIND_CHILD_AT_POSITION, MSG_GEN_FIND_PARENT, MSG_GEN_COUNT_CHILDREN

The following messages provide means of searching the generic tree. It is necessary (in most cases) to check whether a child exists for an object before attempting to delete or access the child. Note that it is bad practice to retrieve optrs using these messages and then some time later send messages manipulating these optrs directly because the tree might change. Instead, use the generic dispatch messages MSG_GEN_CALL_PARENT, MSG_GEN_SEND_TO_CHILDREN, etc.

■ MSG_GEN_FIND_CHILD

```
word      MSG_GEN_FIND_CHILD(
          optr child);
```

This message checks whether the object is a child of the recipient. If so, it returns the child's position. If you wish to find out if a specific child currently exists at a certain position, use MSG_GEN_FIND_CHILD_AT_POSITION.

Source: Unrestricted.

Destination: Any generic object.

Parameters: *child* The optr of child to search for.

Return: The numbered position (zero-based) of the child, or -1 if the specific child is not found.

Interception: Generally not intercepted.

■ MSG_GEN_FIND_CHILD_AT_POSITION

```
optr      MSG_GEN_FIND_CHILD_AT_POSITION(
          word position);
```

This message returns the optr of the child at *position*, if any. Pass the message the zero-based (zero indicates the first child) position to check. If no child is found, the message returns zero.

Source: Unrestricted.

Destination: Any generic object.

Parameters: *position* The position (zero-based) of the child to search for.

Return: The optr of child at position passed, if found. Otherwise returns zero.

Interception: Generally not intercepted.

■ MSG_GEN_FIND_PARENT

```
optr    MSG_GEN_FIND_PARENT();
```

This message returns the parent of the recipient object, if any.

Source: Unrestricted.

Destination: Any generic object.

Return: The optr of the parent object. If no parent is found, this message returns zero.

Interception: Generally not intercepted.

■ MSG_GEN_COUNT_CHILDREN

```
word    MSG_GEN_COUNT_CHILDREN();
```

This message returns the number of children of the recipient.

Source: Unrestricted.

Destination: Any generic object.

Return: The number of children the object has.

Interception: Generally not intercepted.

2.5.2 Manipulating Children Directly

MSG_GEN_ADD_CHILD, MSG_GEN_REMOVE, MSG_GEN_REMOVE_CHILD, MSG_GEN_MOVE_CHILD, MSG_GEN_ADD_CHILD_UPWARD_LINK_ONLY

The following messages create, move, and remove objects set up directly in your **.goc** file or created with **ObjInstantiate()** or **ObjDuplicateResource()**. These objects may have children, in which case those children will travel with their parent (and be moved and destroyed with the parent).

When adding or removing children, you typically have to use a **CompChildFlags** record. This record has two fields, one of which is a dirty flag; the other is a position number indicating a child's position. The record has the following structure:

```
typedef WordFlags CompChildFlags;
#define CCF_MARK_DIRTY  0x8000 /* high bit */
#define CCF_REFERENCE   0x7FFF /* low 15 bits */
#define CCF_REFERENCE_OFFSET  0
/* The CCF_REFERENCE field may have any integral
 * number or may be set to one of the following
 * constants:
 *    CCO_FIRST         first child’s position
 *    CCO_LAST          last child’s position */
```

The **CompChildFlags** fields are

CCF_MARK_DIRTY
If set, this flag indicates that the operation in progress should mark the affected objects dirty. Any objects marked dirty will be saved to the state file upon shutdown.

CCF_REFERENCE
This field consists of the lower 15 bits of the word and is a positive integer representing the position of the child in its parent’s child list. This number cannot be greater than 32767 (0x7fff hex). If the number given is greater than the number of current children, the child will be assumed to be last. For example, a CCF_REFERENCE of five would specify the fourth child of the parent object, or the last child if there are fewer than five children currently. When specifying a position for CCF_REFERENCE, use the CCF_REFERENCE_OFFSET (using the shift operator).

CCO_FIRST and CCO_LAST specify either the first or last child of the parent, respectively. There is no need to use the CCF_REFERENCE_OFFSET in these cases.

■ MSG_GEN_ADD_CHILD

```
void      MSG_GEN_ADD_CHILD(
          optr                child,
          CompChildFlags      flags);
```

This message adds the passed object as a child of the recipient. The child object must not be GS_USABLE before being added to the generic tree. Make sure not to add a child that is already a child of some other parent. It is also illegal to add an object that is already a child of the parent. If necessary, check first if the specific child currently exists using MSG_GEN_FIND_CHILD.

In most cases a routine that adds a new child will follow three phases: checking whether the child exists, adding the child, and setting the child GS_USABLE.

The child object, if already specifically initialized, must be unbuilt before being added to the parent. This ensures that the object will be built out correctly. The internal keyboard search path attribute, GA_KBD_SEARCH_PATH, is also cleared and reset for the child.

Pass this message the optr of the child object to add, along with the **CompChildFlags** to use. If CCF_MARK_DIRTY is specified, the new linkage will be saved to state when the application is detached. You must pass a CCF_REFERENCE in **CompChildFlags** to specify the position to add the child. The special constants CCO_FIRST and CCO_LAST, which are special cases of CCF_REFERENCE, will add the object as the first or last child of the parent, respectively.

Note that the object must currently exist. MSG_GEN_ADD_CHILD merely sets up the correct linkage and reconfigures your UI. Note also that successive additions of children using the flag CCO_FIRST will result in a "reverse order" of children (the last added will be the first child, the first added will be the last).

Source: Unrestricted.

Destination: Any generic object.

Parameters: *child* The optr of the object to add to the current object's children. This child must not be usable.

flags **CompChildFlags** to use when adding the child.

Interception: Generally not intercepted. Custom gadgets may intercept to supplement or supersede default functionality.

Code Display 2-20 ObjDuplicateBlock() with MSG_GEN_ADD_CHILD

```
/* This method duplicates a pre-instantiated version of MyMenu.
 * A duplicated object block may also be added using MSG_GEN_ADD_CHILD.*/
```

```
@method MyProcessClass, MSG_DUPLICATE_MY_MENU {
    MemHandle newBlock;         /* The handle of the Duplicate block. */
    optr newMenu;               /* The optr of the new menu. */
    GeodeHandle procHan;        /* The geode handle of the process. */

    procHan = GeodeGetProcessHandle();
    newBlock = ObjDuplicateBlock(OptrToHandle(@MyMenu), procHan);
                        /* Pass the handle of MyMenu’s resource as well as the
                         * GeodeHandle of the process. */

        /* The new optr is created from the newly created block. */
    newMenu = ConstructOptr(newBlock, OptrToChunk(MyMenu));

        /* Add the duplicated object tree (MyMenu) as the
         * first child of MyPrimary. */
    @call @MyPrimary::MSG_GEN_ADD_CHILD(newMenu, (CCF_MARK_DIRTY | CCO_FIRST));

        /* Then set it usable. Remember, you cannot add a child
         * that is already GS_USABLE. */
    @call @newMenu::MSG_GEN_SET_USABLE(VUM_NOW);
}
```

■ MSG_GEN_ADD_CHILD_UPWARD_LINK_ONLY

```
void    MSG_GEN_ADD_CHILD_UPWARD_LINK_ONLY(
        optr                child);
```

This message sets a parent link to a passed child object without adding a composite link from the parent to the child. This is a “one way” link in that the parent does not have knowledge of its new child. Therefore, it must be used with caution.

Source: Unrestricted.

Destination: Any generic object.

Parameters: *child* optr of the child to add with an upward link.

Interception: Generally not intercepted.

■ MSG_GEN_REMOVE

```
void    MSG_GEN_REMOVE(
        VisUpdateMode       updateMode,
        CompChildFlags      flags)
```

This message removes the receiving object from the generic tree. The object to be removed need not be set not usable.

Source: Anyone.

Destination: Any generic object.

Parameters: *updateMode* Visual update mode. VUM_MANUAL is not allowed.

flags Set CCF_MARK_DIRTY to mark links dirty.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_REMOVE_CHILD

```
void      MSG_GEN_REMOVE_CHILD(
          optr                child,
          CompChildFlags      flags);
```

This message removes the passed object from the recipient. A child must be marked not GS_USABLE in order to be removed. The child must currently exist, so your routine should check this using MSG_GEN_FIND_CHILD.

Pass this message the optr of the child object to be removed along with a word of **CompChildFlags**. If CCF_MARK_DIRTY is specified, the updated linkage will be saved to state when the application is detached.

Source: Unrestricted.

Destination: Any generic object.

Parameters: *child* The optr of child to remove. This child must be non-usable in order to be removed. The child must also exist as a child of the recipient.

flags **CompChildFlags** to use when removing the child.

Interception: Generally not intercepted. Custom gadgets may intercept to supplement or supersede default functionality.

Code Display 2-21 MSG_GEN_REMOVE_CHILD

```
/* This sample method removes the MyChild object from its parent, MyParent. */
@method MyProcessClass, MSG_REMOVE_MY_CHILD {
        /* If the child currently exists, mark it not usable and remove it. */
    if (@call @MyParent::MSG_GEN_FIND_CHILD(@MyChild) != -1) {
        @call @MyChild::MSG_GEN_SET_NOT_USABLE(VUM_NOW);
        @call @MyParent::MSG_GEN_REMOVE_CHILD(@MyChild, CCF_MARK_DIRTY);
    }
}
```

■ MSG_GEN_MOVE_CHILD

2.5

```
void        MSG_GEN_MOVE_CHILD(
            optr                child,
            CompChildFlags      flags);
```

This message moves the given object from the location it currently occupies among its siblings to another location among its siblings. The object will still remain a child of the same parent.

Pass this message the optr of the child to move along with a word of **CompChildFlags**. If you pass the flag CCF_MARK_DIRTY, the new linkage will be saved to state when the application is detached. You must also pass a CCF_REFERENCE so the object will be moved to the position specified. CCO_FIRST and CCO_LAST will move the object to the first or last position, respectively.

If no flags are specified, the object will be moved to the parent object's first position without marking the linkages dirty. Note that for successive moves of children this will result in a "reverse order" of the children.

Note that this message only moves a child among its siblings. To move an object from one parent of the generic tree to another (different) parent, you must use MSG_GEN_REMOVE (or MSG_GEN_REMOVE_CHILD) and MSG_GEN_ADD_CHILD.

Source: Unrestricted.

Destination: Any generic object.

Parameters: *child* The optr of the child to move.

flags **CompChildFlags** to use when moving the child.

Interception: Generally not intercepted.

2.5.3 Branch Construction/Destruction

MSG_GEN_COPY_TREE, MSG_GEN_DESTROY,
MSG_GEN_DESTROY_AND_FREE_BLOCK,
MSG_GEN_BRANCH_REPLACE_PARAMS,
MSG_GEN_BRANCH_REPLACE_OUTPUT_OPTR_CONSTANT

In addition to instantiating and manipulating generic objects one at a time, you can create, move, and destroy entire branches. One way of doing this is copying a resource using **ObjDuplicateResource()** and sending MSG_GEN_ADD_CHILD to the top object. This method has the advantage of retaining the same chunk offsets to objects within the duplicated block as in the source block.

Code Display 2-22 Using ObjDuplicateResource()

```
@start ResourceToCopy;

@object GenItemGroupClass MyList = {
    GI_comp = @AirplaneEntry, @TrainEntry, @CarEntry, @BicycleEntry;
}
@object GenItemClass AirplaneEntry = {}
@object GenItemClass TrainEntry = {}
@object GenItemClass CarEntry = {}
@object GenItemClass BicycleEntry = {}

@end ResourceToCopy;

@method MyProcessClass, MSG_COPY_MY_RESOURCE {
    MemHandle newBlock;

    newBlock = ObjDuplicateResouce(OptrToHandle(@MyList));
    @call @MyPrimary::MSG_GEN_ADD_CHILD(ConstructOptr(newBlock,
                                         OptrToChunk(@MyList));
}
```

Another way to duplicate groups of objects is with MSG_GEN_COPY_TREE. This message greatly simplifies generic tree construction. Those branches of the tree which are roughly similar can be duplicated with one message instead of several.

In many cases, it might be useful to create a UI resource template. This template should contain objects but should not contain object-specific information such as visual monikers, optrs, etc., as this information may be different for different instances of the branch. You can then duplicate these templates with MSG_GEN_COPY_TREE. The trees can then be updated to reflect their unique data either object by object or by using MSG_GEN_BRANCH_REPLACE_PARAMS, which will replace instance data within an entire branch.

2.5

To set up a template, create the generic tree you wish to function as a template, making sure that the top-level object of that tree is not usable (~GS_USABLE). Then use MSG_GEN_COPY_TREE to copy the tree to the proper location. You can then set the other instance data within the method. Finally, to make that tree appear on screen, set the top-level object GS_USABLE. You can remove any tree created with MSG_GEN_COPY_TREE with MSG_GEN_DESTROY. If you use **ObjDuplicateResource()** to copy a resource block, you can send MSG_FREE_DUPLICATE to *any* object in that block to remove it. If the generic branch to be destroyed resides completely within a single block, you may send MSG_GEN_DESTROY_AND_FREE_BLOCK to the top object in that generic branch.

■ MSG_GEN_COPY_TREE

```
optr    MSG_GEN_COPY_TREE(
        MemHandle destBlock,
        ChunkHandle parentChunk,
        word flags);
```

This message copies an entire branch of a generic object tree, starting at the object first called. Pass this message the handle of the destination object block. The object sent the message must not be GS_USABLE.

You may pass *parentChunk* null if you wish the block to remain unattached to the generic tree. Note that if a generic tree is copied using MSG_GEN_COPY_TREE, the objects contained therein not only will have a new resource handle but will also have new chunk handles. Note that this is different from using **ObjDuplicateResource()**, which will keep the same chunk offsets into the object block.

Unlike **ObjDuplicateResource()**, this message will copy the entire branch desired, even if those objects reside in separate resources. This message is therefore more flexible than **ObjDuplicateResource()**, which only copies a given object block.

GenClass provides the default behavior of broadcasting this message down a generic tree, where each object will make a copy of itself in the destination block. It creates an object chunk the same size as the object chunk being copied, copies over the entire contents of any instance fields and creates an outside chunk for a visual moniker, if any.

If you have a subclass of a generic object which references (and therefore might need to create) a chunk outside the object, you must be sure to intercept this message and allocate, copy over, and update any references to this chunk.

Source: Unrestricted.

Destination: Any generic object.

Parameters: *destBlock* The block to copy the tree into. This block cannot be the same as the source object block.

parentChunk The chunk handle of a generic object in the destination block. The recipient of this message will be copied into *parentChunk*. If zero, copy tree into block without linking it to a particular chunk.

flags **CompChildFlags** to use when adding the tree.

Interception: This message is handled by all generic objects. If you wish to intercept this message, you should first call the superclass to copy over the **GenClass** instance data and then copy over any lmem chunks that are referenced by instance data of the intercepting class.

Code Display 2-23 MSG_GEN_COPY_TREE

```
/* Create a template resource block. For demonstration purposes, this block will
 * only contain one object, a GenItem. All declared resources must also be
 * declared in an application’s corresponding .gp file. */

@start MyTemplate;
@object GenItemClass TemplateItem = {
    GI_states = @default & ~GS_USABLE;   /* Make sure it is NOT usable. */
}
@end MyTemplate;

/* Start the Interface resource block */
```

```
@start Interface;
@object GenItemGroupClass MyList = { }
                                /* Object the template above will be added to. */
@end Interface

@method MyProcessClass, MSG_COPY_A_LIST_ITEM {
    optr        newListEntry;

        /* This call copies TemplateItem to the location at MyList. The macros
         * OptrToHandle and OptrToChunk are used to cast the optr of MyList into
         * the proper type for MSG_GEN_COPY_TREE. This new linkage will be marked
         * dirty, and the child will become the first child of MyList. */
    newListEntry = @call @TemplateItem::MSG_GEN_COPY_TREE(
                                OptrToHandle(@MyList),
                                OptrToChunk(@MyList),
                                (CCF_MARK_DIRTY | CCO_FIRST));

        /* The newListEntry is set usable to bring it up on screen. */
    @call @newListEntry::MSG_GEN_SET_USABLE(VUM_NOW);
}
```

■ MSG_GEN_DESTROY

```
void    MSG_GEN_DESTROY(
        VisUpdateMode       updateMode,
        word                flags);
```

This message destroys a generic branch starting at the object called. The message sets all required generic objects not usable, removes them from the generic tree, and destroys them, including any chunks associated with the specific object destroyed. This message is the safest way to destroy a generic branch.

The message will leave the block that the objects previously resided in intact (except for objects removed, of course), but any chunks coming from those resources will be marked dirty and will be resized to zero.

It is usually a good idea to destroy generic branches that are not needed to be saved to state when an application detaches or exits. This ensures that when an application is reattached, it will not contain the previous links. This keeps the state file more compact and avoids the possibility of a child not being saved to state while retaining the parent's child pointer.

The only valid flag in **CompChildFlags** is CCF_MARK_DIRTY, which will mark the links as dirty and save them to state.

Source: Unrestricted.

Destination: Any generic object.

Parameters: *updateMode* **VisUpdateMode** to determine when the generic tree is to be visually updated. May not be VUM_MANUAL.

flags **CompChildFlags** to use when destroying the branch (either CCF_MARK_DIRTY or zero).

Interception: This message is handled by all generic objects. If you wish to intercept this message, you should first destroy any chunks that are referenced by instance data of the intercepting class and then call the superclass to destroy the object's **GenClass** instance data.

■ MSG_GEN_DESTROY_AND_FREE_BLOCK

```
Boolean  MSG_GEN_DESTROY_AND_FREE_BLOCK();
```

This is a utility message used to destroy a generic branch which resides completely within one block. The block will be freed. The object called with this message must be the only top object in the block. This message is called by **UserDestroyDialog()**.

If any object within the block resides on the application active list, or if the object is not a window, a slow, non-optimized approach is taken: the branch is set not usable, removed from the generic tree, and sent MSG_META_BLOCK_FREE. If no object of the block can be found on this list, an optimized approach is taken: the window is taken off the screen; FTVMC, mouse, etc. exclusives released; the linkage changed to be one-way upward; and the block sent MSG_META_BLOCK_FREE. In other words, the whole unbuild process is carefully avoided.

Source: Unrestricted

Destination: Top object within a block to be freed.

Return: *true* if the message was handled, *false* otherwise.

Interception: Generally not intercepted.

■ MSG_GEN_BRANCH_REPLACE_PARAMS

```
void      MSG_GEN_BRANCH_REPLACE_PARAMS(@stack
          BranchReplaceParamType     type,
          dword                      replaceParam,
          dword                      searchParam);
```

This message travels down a generic branch to all of an object's children; the message replaces all instance data specified in *type* with replacement instance data. A typical way to implement this is to set up a template UI branch with MSG_GEN_COPY_TREE. Most often, you will use this message to search and replace optrs set within your template. This is easily done by setting these optrs to certain constant values and then searching for those values in *type*. If you only wish to replace your destination optrs, use MSG_GEN_BRANCH_REPLACE_OUTPUT_OPTR_CONSTANT, which is just a specialized case of this message.

When replacing optrs, the constant BRPT_DUMMY_OPTR_START should be used as the *searchParam* value. If multiple optrs are to be replaced, enumerated values based on that value should be used for other optrs. These values will be constants, and since the **TravelOption** types are also constants, care must be taken to replace these dummy values before anything else is done on the generic branch; otherwise, their output may be interpreted as a **TravelOption**.

This message may not be supported by several subclasses of **GenClass**.

Source: Unrestricted.

Destination: Any generic object.

Parameters: *searchParam* Specifies the search parameter. Typically this search parameter is a pre-set constant used to identify the instance data to replace. The search parameter must be a dword.

replaceParam Specifies the replacement parameter. Any instance data in the generic branch that matches *searchParam* will be replaced with the data specified in *replaceParam*. The data must be a dword.

type A value of **BranchReplaceParamType**. The only value supported is BRPT_OUTPUT_OPTR, which specifies that all optrs with values matching *searchParam* should be replaced by the value in *replaceParam*.

Interception: This message is handled by most generic objects (although not all). If you wish to intercept this message, you should first call the superclass to search and replace any **GenClass** instance data, if applicable, and then search and replace any instance data in the the intercepting class.

■ MSG_GEN_BRANCH_REPLACE_OUTPUT_OPTR_CONSTANT

```
void      MSG_GEN_BRANCH_OUTPUT_OPTR_CONSTANT(
          optr    replacementOptr,
          word    searchConstant);
```

This message is a special case of MSG_GEN_BRANCH_REPLACE_PARAMS where the **BranchReplaceParamType** is BRPT_OUTPUT_OPTR with a search parameter of a constant value set up beforehand. You must pass this message the optr to replace the search constant found.

Source: Unrestricted.

Destination: Any generic object.

Parameters: *replacementOptr* The optr to replace all instance fields with destinations matching the *searchConstant*.

searchConstant The constant to search for and replace with the *replacementOptr*.

Interception: Generally not intercepted. This message invokes MSG_GEN_BRANCH_REPLACE_PARAMS so intercept that instead.

2.6 Vardata

Within **GenClass**, much of your instance data will appear within the basic instance fields allocated with **@instance**. These instance data fields are fixed; each generic object you create will contain each one of these instance fields. You may also make use of special instance data (variable data, also called “vardata”) using the object variable data storage mechanism. See “GEOS Programming,” Chapter 5 of the Concepts Book for complete information on variable data.

Vardata performs many of the same functions that fixed instance data does. However, vardata differs significantly from fixed instance data in that it will only occupy memory if it is in fact declared. You may also only access vardata entries through special kernel routines and messages. For generic objects, most vardata is either in the form of attributes (beginning with ATTR_) or hints (beginning with HINT_).

2.6.1 Optional Attributes (ATTRs)

Vardata attribute entries begin with the form ATTR_*attributename* (as in ATTR_GEN_DESTINATION_CLASS). You can initialize this data within your Goc object declaration. ATTRs within **GenClass** are available for use by any generic object. Several subclasses of **GenClass** also contain their own ATTR vardata entries. In **GenClass** no vardata attributes or hints are set by default.

ATTRs often behave as additional instance data that just happens to be vardata (and therefore optionally added). For generic objects, these should always be considered part of the generic state of an object and therefore may be saved out to the state file. You may, of course, decide not to save any such fields to the state file.

Temporary vardata to be used by an object class can be specified using the format TEMP_*tempname*. Any data set up with TEMP in a generic object should not be saved to state and should not be considered part of the API but rather an internal field. In almost all cases, you will not need to access such vardata entries.

For information on manipulating the variable length instance fields dynamically, along with a list of valid hints for **GenClass**, see “Dynamically Managing VarData” on page 249.

2.6.1.1 Destination Classes

```
ATTR_GEN_DESTINATION_CLASS
```

ATTR_GEN_DESTINATION_CLASS specifies the object class that should handle messages sent out by this object. Typically, an object marked with this ATTR will not have a specific destination optr, but instead a destination path defined by a **TravelOption**. The message for a generic object with a destination class travels down the generic tree along the path specified in the **TravelOption** until it encounters an object class of type ATTR_GEN_DESTINATION_CLASS; at that point, the message will be handled.

ATTR_GEN_DESTINATION_CLASS takes an argument of type **DestinationClassArgs**. This type contains a pointer to a **ClassStruct**. You may have to cast the class name into a **ClassStruct** type when setting ATTR_GEN_DESTINATION_CLASS.

```
typedef struct {
      ClassStruct * DCA_class;
} DestinationClassArgs;
```

Code Display 2-24 Using ATTR_GEN_DESTINATION_CLASS

```
/* This object will send MSG_META_CUT to the application object and follow the
 * target path until it encounters a VisTextClass object. At that point, the
 * message will be handled. */

@object GenTriggerClass MyTrigger = {
    GI_visMoniker = “This trigger cuts text from the target object.”
    GTI_actionMsg = MSG_META_CUT;
    GTI_destination = (TO_APP_TARGET);
/* The class pointer points to a ClassStruct structure. */
    ATTR_GEN_DESTINATION_CLASS = { (ClassStruct *)&VisTextClass };
}

/* To set up a destination class as any generic object, set
 * ATTR_GEN_DESTINATION_CLASS to GenClass. This interaction sends
 * MSG_GEN_SET_NOT_USABLE to the first focus object below the GenInteraction. */

@object GenTriggerClass MyTrigger = {
    GI_visMoniker = “Remove the Focus object”;
    GTI_actionMsg = MSG_GEN_SET_NOT_USABLE;
    GTI_destination = (TO_FOCUS);
    ATTR_GEN_DESTINATION_CLASS = { (ClassStruct *)&GenClass };
}
```

```
typedef enum {
            /* These values may be used as normal
             * TravelOptions, and have been set up
             * so that they will have no value in
             * common with normal TravelOptions. */
      TO_GEN_PARENT=_FIRST_GenClass,
      TO_FOCUS,
      TO_TARGET,
      TO_MODEL,
      TO_APP_FOCUS,
      TO_APP_TARGET,
      TO_APP_MODEL,
      TO_SYS_FOCUS,
      TO_SYS_TARGET,
      TO_SYS_MODEL,
} GenTravelOption;
```

The **TravelOption** types provided with **GenClass** are additional enumerations of the types supplied with **MetaClass**. **MetaClass** provides the types TO_NULL, TO_SELF, TO_OBJ_BLOCK_OUTPUT and TO_PROCESS. Therefore, any generic object can use these types or the further enumerations included here.

These **TravelOption** types typically replace a generic object's destination with a path for a message to be delivered along. Most often, these **TravelOption**s are used in conjunction with the vardata attribute ATTR_GEN_DESTINATION_CLASS. Together, they allow a message to follow a path specified in **TravelOption** until it encounters an object of a class specified in ATTR_GEN_DESTINATION_CLASS. This allows an object to generically send a message down an output path without knowing a specific destination.

2.6.1.2 Initialization File Management

MSG_GEN_LOAD_OPTIONS, MSG_GEN_SAVE_OPTIONS, ATTR_GEN_INIT_FILE_KEY, ATTR_GEN_INIT_FILE_CATEGORY,

ATTR_GEN_INIT_FILE_PROPAGATE_TO_CHILDREN,
ATTR_GEN_USES_HIERARCHICAL_INIT_FILE_CATEGORY

Your object may request information from the GEOS.INI file upon startup. Each object on the appropriate GenApplication GCN list may receive a MSG_GEN_LOAD_OPTIONS upon receiving MSG_META_ATTACH (and a MSG_GEN_SAVE_OPTIONS upon MSG_META_SAVE_OPTIONS). Various generic objects intercept this message and perform actions according to the information provided.

Your object may also access information in the GEOS.INI file as it is being attached by including ATTR_GEN_INIT_FILE_CATEGORY of the GEOS.INI category you are interested in and the ATTR_GEN_INIT_FILE_KEY of the specific keyword you want the data for. If a file key, but no file category, exists within an object declaration, then the system will query the application object for its ATTR_GEN_INIT_FILE_CATEGORY.

Normally only the GenApplication object and any objects in its GCN list will receive MSG_GEN_LOAD_OPTIONS and MSG_GEN_SAVE_OPTIONS. If you wish other children below these objects to also receive these messages (and therefore request or save information from the GEOS.INI file) you can add the attribute ATTR_GEN_INIT_FILE_PROPAGATE_TO_CHILDREN. Any objects with this attribute will send these messages on down the generic tree to their children. Any children may then also contain this attribute and pass the messages on down to their children, etc.

Normally, the object itself and then the GenApplication object is queried for an ATTR_GEN_INIT_FILE_CATEGORY. If instead, the object wants a generic upward query of the object's parents, you can attach ATTR_GEN_USES_HIERARCHICAL_INIT_FILE_CATEGORY to the object. In this case, the object will be queried for its file category, and then query up the generic tree for the first object encountered with an ATTR_GEN_INIT_FILE_CATEGORY. In this manner, one parent may contain a file category and several children may contain different file keys within that category.

Because this method is recursive, it is slow and therefore should be avoided. Most applications should only need one init file category.

■ MSG_GEN_LOAD_OPTIONS

```
void        MSG_GEN_LOAD_OPTIONS(
            GenOptionsParams    *params);
```

This message instructs the generic object to load a value from the GEOS.INI file. It is called automatically by the handler in **GenClass** for MSG_META_LOAD_OPTIONS. The message scans the object's vardata for an ATTR_GEN_INIT_FILE_KEY and an ATTR_GEN_INIT_FILE_CATEGORY to scan the .INI file. If no file category is found in the object's vardata, the handler will search up the generic tree for a parent with an ATTR_GEN_INIT_FILE_CATEGORY.

You may send this message, passing it the proper **GenOptionsParams**. The return value must be cast into the proper type of whatever the entry within the file key is. This is a null-terminated text string.

Source: Sent by **GenClass** on MSG_META_ATTACH. **GenClass** generates the **GenOptionsParams** structure by looking in the object's vardata (and by querying up the generic tree, if necessary).

Destination: Any generic object.

Parameters: *params* **GenOptionsParams** structure (automatically generated by **GenClass**). See MSG_GEN_SAVE_OPTIONS.

Return: Return values must be cast into the proper type of whatever the entry of category and file key is.

Interception: Various generic classes intercept this and provide default behavior

■ MSG_GEN_SAVE_OPTIONS

```
void        MSG_GEN_SAVE_OPTIONS(
            GenOptionsParams    *params);
```

This message instructs the generic object to save its options to the initialization file.

Source: Sent by **GenClass** on MSG_META_SAVE_OPTIONS. **GenClass** generates the proper **GenOptionsParams** structure by looking in the object's vardata (and querying up the generic tree, if necessary).

Destination: Any generic object.

Parameters: *params* **GenOptionsParams** structure (automatically generated by **GenClass**).

Return: Return values must be cast into the proper type of whatever the entry of category and file key is.

Interception: Various generic classes intercept this and provide default behavior

Structures: The **GenOptionsParams** structure is defined as follows:

```
typedef struct {
      char  GOP_category[INIT_CATEGORY_BUFFER_SIZE];
      char  GOP_key[INIT_CATEGORY_BUFFER_SIZE];
} GenOptionsParams;
```

2.6.1.3 Altering Default Geometry Management

ATTR_GEN_POSITION, ATTR_GEN_POSITION_X, ATTR_GEN_POSITION_Y

You may wish, in rare cases, to hard-wire a generic object's position relative to its parent. You can do this by including one of these attributes with a position offset from the parent. These attributes supersede the geometry manager; because of this, you should exercise extreme caution whenever using these attributes. You should only use these attributes in the rare cases when the geometry manager (and hints) cannot accommodate your geometry concerns.

ATTR_GEN_POSITION takes a **Point** argument, specifying the x and y offset (in points) from the parent's top left corner. This point will become the object's top left corner.

ATTR_GEN_POSITION_X specifies the horizontal (x) offset (in points) from the left edge of the parent. The vertical (y) offset will be left to the specific UI (or ATTR_GEN_POSITION_Y) to determine.

ATTR_GEN_POSITION_Y specifies the vertical (y) offset (in points) from the top edge of the parent. The horizontal (x) offset will be left to the specific UI (or ATTR_GEN_POSITION_X) to determine.

Several hints also accomplish similar geometry overrides. HINT_ALIGN_TOP_EDGE_WITH_OBJECT aligns an object's top edge with the top edge of the object (optr) this hint is set to. Similarly, HINT_ALIGN_LEFT_EDGE_WITH_OBJECT, HINT_ALIGN_RIGHT_EDGE_WITH_OBJECT, and HINT_ALIGN_BOTTOM_EDGE_WITH_OBJECT all line up an object's edge with the same edge of the object these hints are set to. One of these hints can be used in each direction. The same concerns over superceding the default geometry manager in the above attributes are valid here as well. Therefore, avoid using these hints if possible.

2.6.1.4 Altering Delayed Mode Activity

```
ATTR_GEN_SEND_APPLY_MSG_ON_APPLY_EVEN_IF_NOT_MODIFIED,
ATTR_GEN_SEND_APPLY_MSG_ON_APPLY_EVEN_IF_NOT_ENABLED
```

Most generic objects act on changes made by the user. Some generic objects may not immediately react to these changes; instead these objects allow the user to select a variety of settings and then later “apply” these changes. These generic objects operate in *delayed mode* and each has an “apply message” associated with it. For these objects, the user may alter the state of the object or its children, but the application only sends out the action to perform those changes when the “apply” message is sent out.

For example, a dialog box may contain a list of settings that represent the state of a paragraph in a word processor. If the user ever actually changes those settings, the dialog box is marked as modified, but the changes are not actually made until the object receives notification to apply its changes (send out its apply message).

By default, these generic objects check whether their state has been modified since the last apply message has been sent out. If their state has not been modified and they receive a request to apply changes made, they instead will ignore the request. Likewise, objects set not enabled will ignore the request. ATTR_GEN_SEND_APPLY_MSG_ON_APPLY_EVEN_IF_NOT_MODIFIED and ATTR_GEN_SEND_APPLY_MSG_ON_APPLY_EVEN_IF_NOT_ENABLED override this behavior, telling the generic object to send out its apply message regardless of the modified or enabled state of a generic object.

In the above example, this would mean that whenever the dialog box receives notification to apply its changes it would not check whether its settings have been altered since the last time it applied its changes. Instead it would apply these changes anyway.

2.6.1.5 Notification of Visibility

```
ATTR_GEN_VISIBILITY_DATA, ATTR_GEN_VISIBILITY_MESSAGE,
ATTR_GEN_VISIBILITY_DESTINATION
```

If a generic object has its GA_NOTIFY_VISIBILITY bit set, it will notify the GenApplication object when it becomes visible or not visible. The object does this by sending MSG_GEN_APPLICATION_VISIBILITY_NOTIFICATION to the GenApplication object, passing the sending object's optr as data.

You can alter this default behavior by including one or more attributes. ATTR_GEN_VISIBILITY_MESSAGE specifies a different message than MSG_GEN_APPLICATION_VISIBILITY_NOTIFICATION; ATTR_GEN_VISIBILITY_DESTINATION specifies a different destination to send visibility notification to than the GenApplication object; ATTR_GEN_VISIBILITY_DATA specifies a different word of data to send out than the default data (the optr of the sending object). These attributes will have no effect on objects not marked GA_NOTIFY_VISIBILITY.

2.6.1.6 Generic Properties

```
ATTR_GEN_PROPERTY, ATTR_GEN_NOT_PROPERTY
```

Objects that exhibit properties (List Objects, GenValues, GenTexts, etc.) sometimes do not exhibit immediate effects when the user interacts with the object. These objects act in what is known as *delayed mode*; they only apply their properties when their object receives a MSG_GEN_APPLY.

Properties objects such as these are usually placed within a GIT_PROPERTIES GenInteraction object. The GIT_PROPERTIES dialog box provides an "Apply" trigger to get the objects to apply their properties. You can, however, change this default behavior, though it is uncommon to need to do so.

ATTR_GEN_PROPERTY, when added to an object that exhibits properties, allows an object to act like it is within a properties dialog box even if it normally operates in immediate mode. That is, it will not apply its changes until explicitly told to do so (using MSG_GEN_APPLY).

ATTR_GEN_NOT_PROPERTY allows an object within a GIT_PROPERTIES dialog box to act like it is not within such a dialog box. That is, any user interaction with the object will result in an immediate application of those changes.

2.6.1.7 Window Management

GenClass supplies several optional attributes that affect the display of windows within the user interface. Windowed objects include GenPrimarys, GenDisplays, and GenInteractions (as dialog boxes).

The optional attributes that follow are divided into two groups. Window layering affects how windowed objects are arranged on the screen; i.e., where they appear and whether they appear above or below other windowed objects. In general, this layering behavior is only determined for windowed objects having the same parent, i.e., windows having the same parent can be custom layered, and their parents will be layered among its siblings.

There also exist optional attributes to alter the manner in which input travels within a window. These optional attributes allow the application to provide floating keyboards for pen input.

Window Layering

```
ATTR_GEN_WINDOW_CUSTOM_LAYER_ID,
ATTR_GEN_WINDOW_CUSTOM_WINDOW_PRIORITY,
ATTR_GEN_WINDOW_CUSTOM_LAYER_PRIORITY,
ATTR_GEN_WINDOW_CUSTOM_PARENT,
ATTR_GEN_WINDOW_CUSTOM_WINDOW
```

ATTR_GEN_WINDOW_CUSTOM_LAYER_ID sets a unique layer identifier for the windowed object. All objects with a common identifier are considered part of the same window layer. These layers can then be manipulated as a unit. By convention, these IDs are usually set to the handles owned by the application; this avoids collisions with other application window layer IDs.

If a null layer ID is specified, the handle of the block the object lies in will be selected as the layer ID at run-time. This is the common method of assigning layer IDs.

ATTR_GEN_WINDOW_CUSTOM_LAYER_PRIORITY allows a windowed object to set its layer priority to a custom value. The optional attribute expects an argument of type **LayerPriority**. Note that all windowed objects with the same Layer ID *must* have the same layer priority; therefore, this attribute must be used with care. It should only be used if a layer ID declared with ATTR_GEN_WINDOW_CUSTOM_LAYER_ID is also used.

ATTR_GEN_WINDOW_CUSTOM_WINDOW_PRIORITY sets a custom window priority for this specific window among all other windows of the same layer ID. This optional attribute expects an argument of type **WindowPriority**.

ATTR_GEN_WINDOW_CUSTOM_PARENT allows windowed objects to set a custom parent window. Because window layering applies only to the ordering of windows having the same parent, a window with a custom parent will have its layer ID, layer priority, and window priority compared to other windows having the same parent as ATTR_GEN_WINDOW_CUSTOM_PARENT.

If ATTR_GEN_WINDOW_CUSTOM_PARENT is set null, this indicates that the system screen window should be the parent. Because this window is the parent window for all base windows in the system, an object setting the screen window as its custom parent will be able to appear on top of (or below) all other windows in the system (that do not have a matching layer ID). This is very low-level behavior and should be used with caution, however.

Window Input Management

```
ATTR_GEN_WINDOW_ACCEPT_INK_EVEN_IF_NOT_FOCUSED,
ATTR_GEN_WINDOW_KBD_OVERRIDE,
ATTR_GEN_WINDOW_KBD_POSITION, MSG_GEN_SET_KBD_OVERRIDE,
MSG_GEN_SET_KBD_POSITION
```

ATTR_GEN_WINDOW_ACCEPT_INK_EVEN_IF_NOT_FOCUSED indicates that the object should accept ink even if it doesn't have the focus. In general, presses on non-focused windows will never turn into ink. If this attribute is present, then presses on the window can turn into ink. The normal mechanism for determining if a press should be ink is then followed.

ATTR_GEN_WINDOW_KBD_OVERRIDE indicates the manner in which a floating keyboard is provided for a given window. This optional attribute expects an argument of type of **KeyboardOverride**. This attribute specifies whether a floating keyboard is provided for text input via a pen device.

The following **KeyboardOverride** values are valid:

KO_NO_KEYBOARD
This type indicates that the window should act as if none of its children accepts text input; no floating keyboard will be made available.

KO_KEYBOARD_REQUIRED
This type indicates that the window should act as if one of its children requires text input; a floating keyboard will be brought on-screen whenever the window gains the focus.

KO_KEYBOARD_EMBEDDED
This type indicates that the application is directly providing the keyboard; no system-created floating keyboard will be provided.

You can change the **KeyboardOverride** exhibited by a windowed object by sending the object MSG_GEN_SET_KBD_OVERRIDE.

The following rules will ensure that a keyboard will be present whenever the user interacts with a display requiring a keyboard: Set the KO_KEYBOARD_REQUIRED bit for any GenDisplay which should be accompanied by a floating keyboard. If all GenDisplays in your geode require a floating keyboard, then set this bit for each of them. GenDisplays that contain embedded keyboards should have the KO_KEYBOARD_EMBEDDED bit set instead of the KO_KEYBOARD_REQUIRED bit.

If you want a keyboard to appear visible even when no GenDisplays are open, mark the GenPrimary KO_KEYBOARD_REQUIRED and mark each GenDisplay KO_NO_KEYBOARD.

ATTR_GEN_WINDOW_KBD_POSITION specifies the default position for the floating keyboard to appear when the window gains the focus. If the user moves the floating keyboard, this attribute is updated to reflect the new position and the keyboard will appear at that position when brought up in the future.

You can change this keyboard position by sending the object MSG_GEN_SET_KBD_POSITION, passing it the **Point** to position the keyboard at.

■ MSG_GEN_SET_KBD_OVERRIDE

```
void        MSG_GEN_SET_KBD_OVERRIDE(
            KeyboardOverride    override);
```

This message sets an object's ATTR_GEN_WINDOW_KBD_OVERRIDE to the passed **KeyboardOverride** value.

Source: Unrestricted.

Destination: Any windowed generic object.

Parameters: *override* **KeyboardOverride**.

Interception: Generally not intercepted.

■ MSG_GEN_SET_KBD_POSITION

```
void        MSG_GEN_SET_KBD_POSITION(
            sword               xCoord,
            sword               yCoord);
```

This message sets an object's ATTR_GEN_WINDOW_KBD_POSITION to the passed **Point** values.

Source: Unrestricted.

Destination: Any windowed generic object.

Parameters: *xCoord* *X* coordinate to position the keyboard (relative to the window).

yCoord *Y* coordinate to position the keyboard (relative to the window).

Interception: Generally not intercepted.

2.6.1.8 Help Management

The Help system is discussed in full in "Help Object Library," Chapter 15. You should read that chapter if you wish to include help within your application. The information provided here is mostly an overview of that information.

Help Files

ATTR_GEN_HELP_FILE, ATTR_GEN_HELP_TYPE, ATTR_GEN_HELP_FILE_FROM_INIT_FILE, ATTR_GEN_HELP_CONTEXT

ATTR_GEN_HELP_FILE stores the help file associated with this object. Help files are generated by the help editor (a modified GeoWrite) from GeoWrite documents and are located in USERDATA\HELP.If the help system queries this object for a help file and this optional attribute does not exist, it will query up the generic tree for the first object with an ATTR_GEN_HELP_FILE.

An object can use ATTR_GEN_HELP_FILE_FROM_INIT_FILE to specify that its help file should be taken from the GEOS.INI file. If this attribute is used, the help controller will look in the GEOS.INI file for a category of the same name as the application and a key named “helpfile.” Thus, to set the help file to “My Own Help File” for the HelpSamp application, you could add the following to your GEOS.INI file:

```
[HelpSamp]
helpfile = My Own Help File
```

If, however, no object has ATTR_GEN_HELP_FILE_FROM_INIT_FILE, this init file entry will not be noticed by the help controller.

ATTR_GEN_HELP_TYPE stores the **HelpType** associated with this object. If the help system queries this object for a help type and this optional attribute does not exist, it will query up the generic tree for the first object with an ATTR_GEN_HELP_TYPE.

Each help file must have several contexts named within the file. These contexts are set up when the help file is written. When an object brings up help, it should reference the specific context for which the user needs help. This context is stored within the object’s ATTR_GEN_HELP_CONTEXT field. Usually this optional attribute is placed on a windowed GenInteraction (dialog box).

If an object needs to set a different help context for itself at run-time, it can use MSG_META_DELETE_VAR_DATA and MSG_META_ADD_VAR_DATA to alter the contents of the object’s ATTR_GEN_HELP_CONTEXT.

Focus Help

```
ATTR_GEN_FOCUS_HELP, ATTR_GEN_FOCUS_HELP_LIB
```

ATTR_GEN_FOCUS_HELP is currently unsupported. This help mechanism is a simple, scaled down version of the previously mentioned help system. A fixed area at the bottom of the screen on certain displays acts as a help text area. An object with ATTR_GEN_FOCUS_HELP indicates that this text area should display the its help text when this object gains the focus within the application. This field expects an optr pointing to the required text.

ATTR_GEN_FOCUS_HELP_LIB is similar to ATTR_GEN_FOCUS_HELP but only applies to objects exported from a library(e.g., controllers).

2.6.1.9 Default Monikers

```
ATTR_GEN_DEFAULT_MONIKER, GenDefaultMonikerType
```

ATTR_GEN_DEFAULT_MONIKER specifies a default moniker for this object. Default monikers are generally used for gstring monikers that occur several places within the system (i.e., they are stored outside of the application). ATTR_GEN_DEFAULT_MONIKER expects an argument of type **GenDefaultMonikerType**.

```
typedef enum {
      GDMT_LEVEL_0,
      GDMT_LEVEL_1,
      GDMT_LEVEL_2,
      GDMT_LEVEL_3,
      GDMT_HELP,
      GDMT_HELP_PRIMARY
} GenDefaultMonikerType;
```

The GDMT_LEVEL monikers correspond to the gstring monikers used to indicate the current user level. GDMT_HELP corresponds to the default help trigger moniker. GDMT_HELP_PRIMARY corresponds to the special help trigger for a GenPrimary object.

2.6.1.10 Feature Links

ATTR_GEN_FEATURE_LINK

This optional attribute indicates that a feature within a GenControl object maps to multiple generic trees within the child block. If a feature in a GenControl so marked is turned off, for example, the controller will remove the feature from the first generic tree associated with this feature and then will remove the feature from the link to the next generic tree. Controllers and features are discussed more fully in “Generic UI Controllers,” Chapter 12.

2.6.1.11 Generic Paths

ATTR_GEN_PATH_DATA, MSG_GEN_PATH_SET, MSG_GEN_PATH_GET, GenFilePath

GenClass is able to keep track of file path information. **GenClass** does not use this information itself, but maintains it so that various subclasses (e.g., the file selector and document control) will be able to use the same instance fields and messages.

Because handlers for all these messages are implemented in **GenClass**, it is possible to store a path with any generic object, whether the particular object normally does anything with the file system or not.

The path itself is stored in the object’s vardata in the ATTR_GEN_PATH_DATA field. In the absence of such a field, **GenClass** will use SP_TOP as the standard path. Subclasses can change this default by intercepting MSG_META_INITIALIZE_VAR_DATA, filling in the **GenFilePath** structure after it has called the superclass (which will actually allocate the attribute).

The ATTR_GEN_PATH_DATA field contains the following structure;

```
typedef struct {
      DiskHandle  GFP_disk;
      PathName    GFP_path;
} GenFilePath;
```

GFP_disk stores the handle of the disk on which the path resides. This value may be initialized to a **StandardPath** constant. *GFP_path* stores the absolute path to the directory; this path may be a relative path if *GFP_disk* is a **StandardPath** directory.

You can retrieve an object's current path with MSG_GEN_PATH_GET. You can alter an object's path with MSG_GEN_PATH_SET.

■ MSG_GEN_PATH_SET

```
Boolean  MSG_GEN_PATH_SET(
         char                *path,
         DiskHandle          disk);
```

This message sets the path associated with the object. Normally, a complete path name must be passed. Note that this path string should not include a drive specifier, as this is implied by the passed disk handle. If the passed **DiskHandle** is actually a **StandardPath** value, then the path string will be taken to be relative to this standard path.

Source: Unrestricted.

Destination: Generic object which has an associated path.

Parameters: *path* Null-terminated pathname; may not be in the same block as the object receiving this message. (If null, default handler will use root directory of disk handle or directory associated with standard path.)

disk Disk handle of path, or **StandardPath** constant.

Return: Returns *true* if error.

Interception: Generally not intercepted. If the object is specifically grown, the message will be forwarded to the specific class after it has been acted on by **GenClass**.

■ MSG_GEN_PATH_GET

```
Boolean  MSG_GEN_PATH_GET(
         char                *buffer,
         word                bufSize);
```

This message returns a null-terminated complete path (no drive letter—drive is implied by the disk handle) for the object. Note that if the object is within a standard directory, the disk handle will be a **StandardPath** constant and the path will appear relative.

Source: Unrestricted.

Destination: Generic object which has an associated path.

Parameters: *buffer* Pointer to character buffer which will be filled with return value.

bufSize Size of the above buffer.

Return: Returns *true* if error (if the path won't fit in the passed buffer or is invalid).

buffer Null-terminated pathname.

Interception: Generally not intercepted.

■ MSG_GEN_PATH_GET_BLOCK

2.6
```
@alias (MSG_GEN_PATH_GET) MemHandle MSG_GEN_PATH_GET_BLOCK(
          char                *buffer, /* This must be NULL */
          word                bufSize);
```

This message returns the handle of a block containing the path.

Source: Unrestricted.

Destination: Generic object which has an associated path.

Parameters: *buffer* This must be NULL.

bufSize This argument is not used.

Return: Handle of block containing path.

Interception: Generally not intercepted.

■ MSG_GEN_PATH_GET_DISK_HANDLE

```
DiskHandle MSG_GEN_PATH_GET_DISK_HANDLE();
```

This message returns the disk handle for the path bound to the object.

Source: Unrestricted.

Destination: Generic object which has an associated path.

Parameters: None.

Return: Disk handle associated with path, or **StandardPath** if object is associated with a standard directory.

Interception: Generally not intercepted.

2.6.2 Hints to the Specific UI

Hints permit the developer to add suggested behavior to the user interface without requiring the specific UI to implement that functionality. Hints therefore provide the specific UI with less stringent guidelines than other attributes. The specific UI has the option of completely ignoring a hint if it cannot understand it or implement it in the requested fashion.

For example, some hints deal with the spacial arrangement of objects and can be ignored if the specific UI cannot accommodate their requests. Developers should use hints to add suggested user interface behavior throughout their UI. This suggested use, however, should not be crucial to their application.

For example, the hint HINT_LIST_CHECKBOXES has no effect under OSF/Motif because the OSF/Motif specification does not include checkboxes. However, in some other specific UI (OpenLook, for example) that feature may be implemented.

Code Display 2-25 Setting Hints

```
@object GenInteractionClass MyInteraction = {
    GI_comp = @OneTrigger, @TwoTrigger, @ThreeTrigger;

        /* Hints are added directly in an object's declaration. Note that each
         * vardata entry will expand the size of the object in memory. */
    HINT_ORIENT_CHILDREN_HORIZONTALLY;
        /* This hint instructs the specific UI to arrange the object's children
         * in one or more horizontal rows, if possible. */
}

/* Hints are entered separately in C if multiple hints are desired. */

@object MyObjectClass MyObject = {
    GI_comp = @MyObjectChild;
    HINT_LIST_CHECKBOXES;
    HINT_ORIENT_CHILDREN_HORIZONTALLY;
}
```

2.6.2.1 System Attributes

SystemAttrs, HINT_IF_SYSTEM_ATTRS, HINT_ELSE, HINT_ENDIF

The system attributes are global flags that describe what sort of environment is running the generic object. HINT_IF_SYSTEM_ATTRS is used to conditionally add hints to an object based on these **SystemAttrs**. If the

SystemAttrs set in the HINT_IF_SYSTEM_ATTRS field are true for the current system, then the hints that follow (until a HINT_ENDIF is encountered) are included. If no HINT_ENDIF is encountered, then only the next hint is included. If the **SystemAttrs** do not match the current system, the following group of hints is instead deleted.

The following **SystemAttrs** are defined:

SA_TINY If set, the screen must be either horizontally or vertically tiny.

SA_HORIZONTALLY_TINY
If set, the screen must be horizontally tiny.

SA_VERTICALLY_TINY
If set, the screen must be vertically tiny.

SA_COLOR If set, the system must have a color screen.

SA_PEN_BASED
If set, the system must be pen-based.

SA_KEYBOARD_ONLY
If set, the system must be set keyboard-only.

SA_NO_KEYBOARD
If set, the system must be set no-keyboard.

SA_NOT If this flag is set, any other set bits indicate criteria that must *not* match. This is the equivalent of a logical NOT operation.

The HINT_ELSE hint can be used between HINT_IF_SYSTEM_ATTRS and HINT_END_IF. This allows the inclusion of a separate set of hints to be included when the system conditions in HINT_IF_SYSTEM_ATTRS are not satisfied.

Code Display 2-26 System Attributes

```
/* The SA_NOT bit acts as a logical not operation. In this case, the hint
 * HINT_EXPAND_HEIGHT_TO_FIT_PARENT will only be included in the object
 * declaration if the system is not pen-based. In addition, if the system is tiny,
 * the other hints will be included. The HINT_ENDIF marks the end of these
 * conditional hints. */
```

```
    HINT_IF_SYSTEM_ATTRS = SA_NOT | SA_PEN_BASED;
        HINT_EXPAND_HEIGHT_TO_FIT_PARENT;
    HINT_IF_SYSTEM_ATTRS = SA_TINY;
        HINT_MINIMIZE_SIZE;
        HINT_DISPLAY_CURRENT_SELECTION;
    HINT_ENDIF;

/* Example within an object declaration. */

@object GenInteractionClass GroupingObject = {
    GI_comp = @One, @Two, @Three, @Four, @Five, @Six;
    HINT_IF_SYSTEM_ATTRS = SA_HORIZONTALLY_TINY;
        HINT_ORIENT_CHILDREN_VERTICALLY;
        HINT_WRAP_AFTER_CHILD_COUNT = 3;
    HINT_ELSE;
        HINT_ORIENT_CHILDREN_HORIZONTALLY;
    HINT_ENDIF;
}
```

2.6.2.2 Default Actions

HINT_DEFAULT_DEFAULT_ACTION,
HINT_ENSURE_TEMPORARY_DEFAULT,
HINT_PREVENT_DEFAULT_OVERRIDES

Normally, objects in the user interface are activated by direct action (e.g., clicking on them). Objects may also be indirectly activated, however. The object activated indirectly is known as the default action object. For example, within a window, pressing ENTER or sending the window MSG_GEN_ACTIVATE_INTERACTION_DEFAULT will activate the object marked as the default action.

HINT_DEFAULT_DEFAULT_ACTION marks an object as the default action for the window it appears in. This hint is only relevant for GenTriggers or dialog GenInteractions (that are brought up by an activation trigger). The default activation object is usually activated by pressing the ENTER key within the window.

Normally, whenever a trigger is activated within a window, it will become the default activation object the next time around. Even if a trigger has HINT_DEFAULT_DEFAULT_ACTION, if some other trigger is activated, that other trigger will become the default action in the future. To prevent these automatic default overrides, add HINT_PREVENT_DEFAULT_OVERRIDES to the object's instance data.

HINT_ENSURE_TEMPORARY_DEFAULT ensures that an object that can be navigated to (via the TAB key for example) will act as a default activation object even if the specific UI does not normally allow such behavior. (OSF/Motif does this automatically.)

2.6

2.6.2.3 Keyboard Navigation Hints

HINT_NAVIGATION_ID, HINT_NAVIGATION_NEXT_ID

Most specific UIs allow keyboard navigation within windows, usually through use of the TAB key. Normally, the navigation path follows the order of the children within the windowed object, and this is sufficient for most needs.

HINT_NAVIGATION_ID sets a navigation identifier for an object. Objects may "jump" to this object by including HINT_NAVIGATION_NEXT_ID in their instance data with a value equal to the matching navigation ID of the object to travel to.

Code Display 2-27 Navigation IDs

```
/* Essentially, this code allows keyboard navigation to skip the Two
 * trigger. Hitting the TAB key on the ‘One’ trigger navigates the focus to the
 * Three trigger. */

@object GenTriggerClass One = {
    GI_visMoniker = “TAB here to get to Three”;
    HINT_NAVIGATION_NEXT_ID = 3;
}

@object GenTriggerClass Two = {
    GI_visMoniker = “2”;
}
```

```
@object GenTriggerClass Three = {
    GI_visMoniker = “3”;
    HINT_NAVIGATION_ID = 3;
}
```

Note that an object with a matching navigation ID must exist, so this method is not recommended. If possible, your UI should be constructed in such a way to allow the default behavior.

2.6.3 Dynamically Managing VarData

MSG_META_ADD_VAR_DATA, MSG_META_DELETE_VAR_DATA, MSG_GEN_ADD_GEOMETRY_HINT, MSG_GEN_REMOVE_GEOMETRY_HINT

You must use special kernel routines and **MetaClass** messages to access, add, or remove entries within an object’s instance fields.

- **ObjVarScanData()**
  This routine scans an object’s vardata and calls routines listed in the variable data handler table. This routine is useful for processing several HINTs and ATTRs at one time.

- MSG_META_ADD_VAR_DATA
  You can use this message to add a vardata entry to an object. The object must first be set not usable before adding the entry. Set the object usable again after you have added the vardata entry.

- MSG_META_DELETE_VAR_DATA
  Use this message to delete a vardata entry from an object. The object must first be set not usable before removing the entry. Set the object usable again after you have removed the vardata entry.

- **ObjVarFindData()**
  This routine checks the object for the passed type and returns a pointer to its data. The message returns *true* if the entry is present, *false* if it is not. This routine is useful for processing a single vardata entry.

In addition to these routines and messages, two messages specific to **GenClass** allow you to add or remove hints dynamically: MSG_GEN_ADD_GEOMETRY_HINT and MSG_GEN_REMOVE_GEOMETRY_HINT. These messages allow geometry and window positioning hints to be added or removed while the object is on-screen. Sending a GS_USABLE object one of these messages with a valid geometry hint forces that object to redraw itself according to the new geometry.

■ MSG_GEN_ADD_GEOMETRY_HINT

```
void        MSG_GEN_ADD_GEOMETRY_HINT(
            word                hint,
            word                hintArgument,
            VisUpdateMode       updateMode);
```

This message adds a geometry hint to a generic object. The object may be GS_USABLE at the time, in which case the object is redrawn according to the new geometry configuration.

Source: Unrestricted.

Destination: Any generic object.

Parameters:

hint	The hint to add to the object's instance data
hintArgument	A word of data for any hints that require an argument.
updateMode	**VisUpdateMode** to determine when the object will be redrawn.

Interception: Generally not intercepted.

■ MSG_GEN_REMOVE_GEOMETRY_HINT

```
void        MSG_GEN_REMOVE_GEOMETRY_HINT(
            word                hint,
            VisUpdateMode       updateMode);
```

This message removes a geometry hint from an object's instance data. The object may be GS_USABLE at the time, in which case it will be redrawn according to the new geometry configuration.

Source: Unrestricted.

Destination: Any generic object.

Parameters:

hint	The hint to remove.
updateMode	**VisUpdateMode** to determine when the object will be redrawn.

Interception: Generally not intercepted.

◆ Objects

2.7 Special Message Passing

It is often useful to pass messages "generically" rather than having to specify the particular optr directly. For example, in many cases, retrieving and storing the optr of an object is cumbersome, and it would be much easier to say "deliver this message to my parent." **GenClass** provides many useful messages which handle cases such as these. These message passers are of three types:

- Parent and Child Generic Message Passers
- Generic Upward Queries
- Object-Specific Upward Queries

In all cases, the message must be stored as a *classed event*. A classed event is a combination of a class and a message. The message itself may pass data. The classed event enables the object to specify which class it would like the message to be handled by. In Goc, a classed event is created by encapsulating the message and class using the **@record** keyword.

2.7.1 Parent and Child Message Passing

MSG_GEN_CALL_PARENT, MSG_GEN_SEND_TO_PARENT,
MSG_GEN_SEND_TO_CHILDREN

Three messages enable you to pass other messages to a generic object's parents and children without having to know the proper optrs. Using these messages, you can perform operations on any generic object's parents and children. If you need return values from an operation being performed by the parent object, use MSG_GEN_CALL_PARENT. If no return values are needed, you may use MSG_GEN_SEND_TO_PARENT.

To pass a message to all of your generic children, use MSG_GEN_SEND_TO_CHILDREN. Note that no comparable MSG_GEN_CALL_CHILDREN exists because it is meaningless to expect return values from multiple objects for a single event.

You may also use the Goc macros **genParent** and **genChildren** to send a message to a parent or the generic children of an object.

■ MSG_GEN_CALL_PARENT

```
void        MSG_GEN_CALL_PARENT(
            EventHandle         event);
```

This message delivers an event to the generic parent of the recipient. This message must pass a classed event that the parent object will handle. You should use this message if return values are expected. Always make sure to cast the return (following the call) into the proper type. The most effective way to do this is by enclosing the actual message sent within parentheses. The event will be freed after it is sent.

Source: Unrestricted.

Destination: Any generic object.

Parameters: *event* The classed event to deliver to parent of this object.

Return: The return value of the classed event (cast to the proper type).

Interception: Generally not intercepted. Custom gadgets may handle to supplement or supersede default functionality.

Code Display 2-28 MSG_GEN_CALL_PARENT

```
/* The following method retrieves the visual moniker
 * of an object’s parent. */

@method MyProcessClass, MSG_GET_MY_PARENTS_MONIKER {
    ChunkHandle parentMoniker;
    EventHandle myEvent;

        /* Encapsulate the message to be handled
         * by any generic (GenClass) object. */

    myEvent = @record GenClass::MSG_GEN_GET_MONIKER();

/* Calls the parent of EntryNumberTwo with the classed event specified above. Note
 * that the return value is cast to type (MSG_GEN_GET_MONIKER) because
 * MSG_GEN_CALL_PARENT itself returns void. */

    parentMoniker = @call (MSG_GEN_GET_MONIKER)
                        @EntryNumberTwo::MSG_GEN_CALL_PARENT(myEvent);
    return(parentMoniker); /* return the parentMoniker. */
}
```

◆ Objects

■ MSG_GEN_SEND_TO_PARENT

```
void        MSG_GEN_SEND_TO_PARENT(
            EventHandle event);
```

This message sends an encapsulated event to the parent but expects no return values. The event will be freed after it is sent.

Source: Unrestricted.

Destination: Any generic object.

Parameters: *event* The classed event to deliver to parent of this object.

Interception: Generally not intercepted. Custom gadgets may handle to supplement or supersede default functionality.

■ MSG_GEN_SEND_TO_CHILDREN

```
void        MSG_GEN_SEND_TO_CHILDREN(
            EventHandle event);
```

This message sends an encapsulated event to all children of the generic object receiving it. This message cannot return values. The event will be freed after it is sent.

Source: Unrestricted.

Destination: Any generic object.

Parameters: *event* The classed event to deliver to all children.

Interception: Generally not intercepted. Custom gadgets may handle to supplement or supersede default functionality.

2.7.2 Generic Upward Queries

MSG_GEN_GUP_CALL_OBJECT_OF_CLASS,
MSG_GEN_GUP_SEND_TO_OBJECT_OF_CLASS,
MSG_GEN_GUP_TEST_FOR_OBJECT_OF_CLASS,
MSG_GEN_GUP_FIND_OBJECT_OF_CLASS, MSG_GEN_GUP_QUERY,
MSG_GEN_GUP_INTERACTION_COMMAND, MSG_GEN_GUP_FINISH_QUIT

GenClass provides the capability to search up the generic tree beyond the parent. Using the following messages, the UI can continue passing classed events up the generic tree until it reaches an object of the proper class. This behavior is known as a Generic UPward query (GUP).

For example, if you specify **GenInteractionClass** in your classed event and later send a GUP message to any object in the generic tree, the stored message will be handled at the first available GenInteraction object it encounters.

MSG_GEN_GUP_CALL_OBJECT_OF_CLASS performs a GUP, returning values from the passed message. You must cast the return values into the proper type based on the return values of the passed message. MSG_GEN_GUP_SEND_TO_OBJECT_OF_CLASS performs a GUP but does not allow return values.

MSG_GEN_GUP_TEST_FOR_OBJECT_OF_CLASS performs a GUP whose sole function is to search for the existence of an object of the specified class among the object's parents.

MSG_GEN_GUP_FIND_OBJECT_OF_CLASS performs a GUP searching for any object of the passed class, but it also returns the optr of the object found. Note that it is unwise to later use this optr as the generic tree may have changed.

■ MSG_GEN_GUP_CALL_OBJECT_OF_CLASS

```
void      MSG_GEN_GUP_CALL_OBJECT_OF_CLASS(
          EventHandle event);
```

This message performs a generic upward query, passing the classed event upward until it reaches an object of the passed class. This message allows return values and should be cast into whatever return values are expected of the passed message. This is most easily done by enclosing the message sent within parentheses.

An object of the expected class should be present or the event will not be delivered to any object. To check for the existence of such a class, use MSG_GEN_GUP_TEST_FOR_OBJECT_OF_CLASS.

Source: Unrestricted.

Destination: Any generic object.

Parameters: *event* The classed event to deliver to an object up the generic tree.

Return: The return values of the classed event (cast to the proper type).

Interception: Generally not intercepted. Custom gadgets may handle to supplement or supersede default functionality.

■ MSG_GEN_GUP_SEND_TO_OBJECT_OF_CLASS

```
void        MSG_GEN_GUP_SEND_TO_OBJECT_OF_CLASS(
            EventHandle event);
```

This message performs a generic upward query, passing the classed event upward until it reaches an object of the described class. Since this message performs a send, it does not allow return values. An object of the expected class should be present; otherwise, the event will not be delivered to any object. To check for the existence of such a class, use MSG_GEN_GUP_TEST_FOR_OBJECT_OF_CLASS.

Source: Unrestricted.

Destination: Any generic object.

Parameters: *event* The classed event to deliver to an object up the generic tree.

Interception: Generally not intercepted. Custom gadgets may handle to supplement or supersede default functionality.

Code Display 2-29 Nesting Classed Events

```
/* The following method can be used by any object in the generic tree below the
 * primary to set all menus and GenInteractions below the primary window enabled
 * all at once. */

@method MyProcessClass, MSG_ENABLE_ALL_MY_MENUS {
    EventHandle menuEvent, primaryEvent;

/* The first classed event (menuEvent) encapsulates MSG_GEN_SET_ENABLED for any
 * object of GenInteractionClass (typically menus). The @record keyword
 * encapsulates the message to be handled by GenInteractionClass. The
 * VisUpdateMode delays updates via the Application queue to avoid constant visual
 * updates and “flashing.” */

    menuEvent = @record GenInteractionClass::MSG_GEN_SET_ENABLED(
                                    VUM_DELAYED_VIA_APP_QUEUE);

/* The second classed event binds MSG_GEN_SEND_TO_CHILDREN to the GenPrimaryClass.
 * Sending this message to the GenPrimary will send the menuEvent to all
 * its children. In this case, sending this event to the Primary will send
 * MSG_GEN_SET_ENABLED to all its GenInteraction children. */

    primaryEvent = @record GenPrimaryClass::MSG_GEN_SEND_TO_CHILDREN(menuEvent);
/* Finally, to send this message to the GenPrimary from any object below the
 * GenPrimary, simply send a MSG_GEN_GUP_SEND_TO_OBJECT_OF_CLASS with the
 * PrimaryEvent above. The GUP message will send the message
 * MSG_GEN_SEND_TO_CHILDREN up the generic tree until it encounters the GenPrimary,
 * at which point the Primary will send the MenuEvent MSG_GEN_SEND_TO_CHILDREN to
 * all its GenInteraction children. */

    @send @MyObject::MSG_GEN_GUP_SEND_TO_OBJECT_OF_CLASS(primaryEvent);
}
```

■ MSG_GEN_GUP_TEST_FOR_OBJECT_OF_CLASS

```
Boolean   MSG_GEN_GUP_TEST_FOR_OBJECT_OF_CLASS(
          ClassStruct  *class);
```

This message searches up the generic tree for any object of the specific class. This message is useful for testing the existence of required classes before sending MSG_GEN_GUP_CALL_OBJECT_OF_CLASS and MSG_GEN_GUP_SEND_TO_OBJECT_OF_CLASS.

Source: Unrestricted.

Destination: Any generic object.

Parameters: *class* **ClassStruct** to search for among the object's parents.

Return: Will return *true* if the class is found, *false* if not.

Interception: Generally not intercepted. Custom gadgets may handle to supplement or supersede default functionality.

■ MSG_GEN_GUP_FIND_OBJECT_OF_CLASS

```
optr      MSG_GEN_GUP_FIND_OBJECT_OF_CLASS(
          ClassStruct  *class);
```

This message performs a function similar to that of MSG_GEN_GUP_TEST_FOR_OBJECT_OF_CLASS, but it returns the optr of the class found, if any. This optr can then be used as the recipient of other messages. You should not use an optr returned by this message at a later time, as the object may have been removed.

Source: Unrestricted.

Destination: Any generic object.

Parameters: *class* **ClassStruct** to search for.

Return: The optr of the object of the requested class, or a null optr if no such object was found.

Interception: Generally not intercepted. Custom gadgets may handle to supplement or supersede default functionality.

■ MSG_GEN_GUP_QUERY

```
void      MSG_GEN_GUP_QUERY(
          GenReturnParams     *retValue,
          word                queryType);
```

This message is used to query up the generic composite tree.

Source: Anyone.

Destination: Any usable generic object, though only certain objects will answer certain queries.

Parameters: *retValue* Structure to hold return values.

queryType **GenUpwardQueryType**.

The **GenUpwardQueryType** should be one of the following:

GUQT_DELAYED_OPERATION
This query type determines if a gadget should be operating in “delayed mode,” i.e., whether it should wait for a MSG_GEN_APPLY before sending notification to its output, or whether it should send notification whenever it changes state.

GUQT_FIELD
Returns field object in *GRP_cx*:*GRP_dx* and window (if realized) handle in *GRP_bp*. This is a reasonably safe thing to do, since the field window will remain open until all applications within the field are shut down. Only field objects or substitutes for them should intercept this message.

GUQT_SCREEN
Returns screen object in *GRP_cx*:*GRP_dx* and window (if any) handle in *GRP_bp*. This is a reasonably safe thing to do, since the screen window will remain open until all applications within the field are shut down. Screen object being returned will be default screen for the field.

Return: Nothing returned explicitly. *retValue* struct filled with values which depend on query type.

Structures:

```
typedef struct {
      word  GRP_ax;
      word  GRP_bp;
      word  GRP_cx;
      word  GRP_dx;
} GenReturnParams;
```

■ MSG_GEN_GUP_INTERACTION_COMMAND

2.7

```
Boolean  MSG_GEN_GUP_INTERACTION_COMMAND(
         InteractionCommand  command);
```

This message travels up the generic tree until it reaches a non-GIT_ORGANIZATIONAL GenInteraction. Once there, it performs the indicated **InteractionCommand**. See **InteractionCommand** declaration for the different command types and how specific user interfaces interpret the commands. If the GenInteraction is a dialog brought up with **UserDoDialog()**, **InteractionCommand** will be returned to the caller, after performing the command, via MSG_GEN_INTERACTION_RELEASE_BLOCKED_THREAD_WITH_RESPONSE .

Source: Unrestricted. Normally sent by a MSG_GEN_INTERACTION_ACTIVATE_COMMAND handler or by a custom trigger's action message handler to perform some default command after perform some application-specific function.

Destination: Any generic object. Will travel up generic tree, stopping at first non organizational interaction.

Parameters: *command* Standard InteractionCommand.

Return: Returns *true* if handled.

Interception: Not normally intercepted.

■ MSG_GEN_GUP_FINISH_QUIT

```
void     MSG_GEN_GUP_FINISH_QUIT(
         Boolean            abortFlag,
         Boolean            notifyParent);
```

Message to be used by nodes that implement a GCN active list. This should be used if MSG_META_QUIT is handled and sent to the active list items by that node. When a node has finished sending a MSG_META_QUIT to the active list or if the quit is aborted, this allows the notification of the above node in the generic tree.

Source: Sent by active list nodes that implement quit mechanism.

Destination: Any generic object involved in quit mechanism.

Parameters: *abortFlag* Pass true to abort. This will be passed on to MSG_META_FINISH_QUIT.

notifyParent Pass true to notify parent of finished quit sequence.

Interception: Generally not intercepted. Default handler sends MSG_META_FINISH_QUIT to self, then sends to generic parent if flag indicates such. MSG_META_FINISH_QUIT is normally the message to intercept to know when a quit is aborted or finished.

2.7.3 Object-Specific Queries

```
MSG_GEN_CALL_APPLICATION, MSG_GEN_SEND_TO_PROCESS,
MSG_GEN_CALL_SYSTEM,MSG_GEN_OUTPUT_ACTION
MSG_GEN_QUERY_NEED_TO_BE_ON_ACTIVE_LIST
```

Besides the Generic Upward Queries, **GenClass** also contains several object-specific queries. These messages behave in the same manner as the GUP messages except that there is no need to set up a classed event. The object-specific upward query automatically sets up the class to send the message to.

MSG_GEN_CALL_APPLICATION sends a message up the generic tree to the GenApplication object. This message allows return values and should be cast to whatever return values are expected of the passed message.

MSG_GEN_SEND_TO_PROCESS sends a message to the process object associated with this object. Because this message may cross threads, no return values are allowed.

MSG_GEN_CALL_SYSTEM sends a message up the generic tree to the GenSystem object. This message allows return values and should be cast into whatever return values are expected of the passed message.

MSG_GEN_QUERY_NEED_TO_BE_ON_ACTIVE_LIST

■ MSG_GEN_CALL_APPLICATION

```
void        MSG_GEN_CALL_APPLICATION(
            EventHandle event);
```

This message calls the GenApplication object associated with the recipient. This message allows return values and should be cast to whatever return values are expected of the passed message.

Source: Unrestricted.

Destination: Any generic object.

Parameters: *event* The classed event to deliver to the object's GenApplication object.

Return: The return values of the classed event (cast to the proper type).

Interception: Generally not intercepted. Custom gadgets may handle to supplement or supersede default functionality.

■ MSG_GEN_SEND_TO_PROCESS

```
void        MSG_GEN_SEND_TO_PROCESS(
            EventHandle event);
```

This message sends the event to the Process object of your application. Because the message may cross threads, return values are not allowed.

Source: Unrestricted.

Destination: Any generic object.

Parameters: *event* The classed event to deliver to the object's GenProcess object.

Interception: Generally not intercepted. Custom gadgets may handle to supplement or supersede default functionality.

■ MSG_GEN_CALL_SYSTEM

```
void        MSG_GEN_CALL_SYSTEM(
            EventHandle event);
```

This message calls the GenSystem object (the root object) associated with your application and sends the event. If you expect return values, make sure to cast them into the proper variable type.

Source: Unrestricted.

Destination: Any generic object.

◆ Objects

Parameters: *event* The classed event to deliver to the object's GenSystem object.

Return: The return values of the classed event (cast to the proper type).

Interception: Generally not intercepted. Custom gadgets may handle to supplement or supersede default functionality.

■ MSG_GEN_OUTPUT_ACTION

```
void    MSG_GEN_OUTPUT_ACTION(
        EventHandle         event,
        optr                dest);
```

This message sends an event via the destination optr (if any) of the object sent the message. The default handler (which is rarely subclassed) also scans the vardata looking for an ATTR_GEN_DESTINATION_CLASS and records that class in a classed event.

This message is usually sent at a low level and dispatches a message via the destination optr of the object. You may intercept MSG_GEN_OUTPUT_ACTION if you wish knowledge of when an object is being activated (although you may wish to intercept MSG_GEN_ACTIVATE in those cases). If you do send this message yourself, you must either pass a destination optr or leave it null, depending on the object being activated.

Source: Normally sent by an object to itself.

Destination: Normally sent by an object to itself.

Parameters: *event* A classed event.

optr The destination optr to send event to or a **TravelOption**.

Interception: Generally not intercepted.

■ MSG_GEN_QUERY_NEED_TO_BE_ON_ACTIVE_LIST

```
Boolean MSG_GEN_QUERY_NEED_TO_BE_ON_ACTIVE_LIST();
```

This message queries whether an object still needs to be on an active list. The active list mechanism sends this message if a mechanism no longer needs an object to remain on the active list; it checks whether any other mechanism also needs the object to be on the active list also. Windowed specific UI objects, for example, will return *true* if ATTR_INITIATE_ON_STARTUP is present on the object, because this optional attribute requires the object to be on the active list.

Source: Unrestricted, though it is generally called on itself. Specifically, if an object no longer needs to remain on the active list for a particular reason, this message should be called on the object to see if it should remain on the active list for any other reason. If so, the caller should not remove the object from the active list. The only exception to this would be if the object were being destroyed, in which the object should be removed in any case.

Destination: Any generic object.

2.8

Return: *true* if the object needs to remain on the active list for some other reason.

Interception: Object classes which add the object to the active list must intercept this message and return *true* if the class desires the object to remain on the active list. If the message is intercepted and this is not the case, it should be sent to the superclass for handling.

2.8 Visual Refreshing

MSG_GEN_UPDATE_VISUAL

MSG_GEN_UPDATE_VISUAL performs a visual update of the object sent the message. Use this message to force a visual update on objects previously updated using the **VisUpdateMode** VUM_MANUAL.

■ MSG_GEN_UPDATE_VISUAL

```
void    MSG_GEN_UPDATE_VISUAL(
        VisUpdateMode updateMode);
```

This message updates the visual tree at the node specified and is used for updating objects modified with VUM_MANUAL.

Source: Unrestricted.

Destination: Any generic object.

Parameters: *updateMode* **VisUpdateMode** to determine when the object should be redrawn.

Interception: Generally not intercepted.

◆ Objects

2.9 Setting Sizes

MSG_GEN_SET_INITIAL_SIZE, MSG_GEN_SET_MINIMUM_SIZE, MSG_GEN_SET_MAXIMUM_SIZE, MSG_GEN_SET_FIXED_SIZE, MSG_GEN_GET_INITIAL_SIZE, MSG_GEN_GET_MINIMUM_SIZE, MSG_GEN_GET_MAXIMUM_SIZE, MSG_GEN_GET_FIZED_SIZE, MSG_GEN_RESET_TO_INITIAL_SIZE, CompSizeHintArgs

2.9

These messages all perform resizing operations on generic objects. In some cases, the geometry is not redone by sending these messages; the object will need to be set not usable and then set usable to force the new sizing scheme. In most cases, however, sending these messages will affect the current geometry of an object. These messages manipulate their associated hints, either by creating new hints or by altering old ones. As such, these messages will only affect an object if the associated hint does also.

These messages alter four size restrictions associated with an object: initial, maximum, minimum, and fixed. These restrictions are set up using the **GenClass** hints HINT_INITIAL_SIZE, HINT_MAXIMUM_SIZE, HINT_MINIMUM_SIZE, and HINT_FIXED_SIZE, all of which are described in “Managing UI Geometry,” Chapter 12 of the Concepts Book.

The size parameter required by each of these hints is a structure of type **CompSizeHintArgs**. This structure is shown below:

```
typedef struct {
    SpecWidth     CSHA_width;
    SpecHeight    CSHA_height;
    sword         CSHA_count;
} CompSizeHintArgs;
```

Many of the messages pass a structure of **GetSizeArgs** to place size information retrieved from the message. This size information is used by the specific UI to visually construct the object, bypassing default sizes in the process. The structure is shown below:

```
typedef struct {
    word    GSA_width;
    word    GSA_height;
    word    GSA_unused;
    word    GSA_count;
} GetSizeArgs;
```

■ MSG_GEN_GET_INITIAL_SIZE

2.9

```
void    MSG_GEN_GET_INITIAL_SIZE(
        GetSizeArgs *initSize);
```

This message retrieves the initial size specifications of a generic object stored in the hint HINT_INITIAL_SIZE. The object must be passed a pointer to a structure of type **GetSizeArgs** to store the retrieved data. If the hint HINT_INITIAL_SIZE does not exist for this object, the structure passed to the object of size **GetSizeArgs** will remain empty.

Source: Unrestricted.

Destination: Any generic object.

Parameters: *initSize* A pointer to an empty **GetSizeArgs** structure.

Return: The **GetSizeArgs** structure will contain the object's initial size.

Interception: Generally not intercepted.

■ MSG_GEN_SET_INITIAL_SIZE

```
void    MSG_GEN_SET_INITIAL_SIZE(@stack
        byte                updateMode,
        word                count,
        word                height,      /* SpecHeight */
        word                width);      /* SpecWidth */
```

This message sets the initial size of a generic object (its size on being first built). This is only done by creating or modifying the hint HINT_INITIAL_SIZE; there is no guarantee that the specific UI will implement this message in the expected fashion.

The message must pass a **VisUpdateMode**. This message will only affect the size of an object when it first opens; it will not modify the geometry of a currently built object. For the defined geometry to take effect on an already built object, the object must be set not usable and then set usable again.

Source: Unrestricted.

◆ Objects

Destination: Any generic object.

Parameters: *count* The number of children (or zero, if not applicable).

height The height of each child.

width The width of the composite.

Interception: Generally not intercepted.

■ MSG_GEN_RESET_TO_INITIAL_SIZE

```
void    MSG_GEN_RESET_TO_INITIAL_SIZE(
        VisUpdateMode       updateMode);
```

This message forces the recipient and all of its children to be resized according to their initial size specifications. If the objects contain HINT_INITIAL_SIZE, they will be resized according to the hint; otherwise they will be resized according to their default geometry. This message is useful for activating sizes set with MSG_GEN_SET_INITIAL_SIZE.

Source: Unrestricted.

Destination: Any GS_USABLE generic object.

Parameters: *updateMode* **VisUpdateMode** to determine when the object will be redrawn.

Interception: Generally not intercepted.

■ MSG_GEN_GET_MINIMUM_SIZE

```
void    MSG_GEN_GET_MINIMUM_SIZE(
        GetSizeArgs  *minSize);
```

This message retrieves the recipient's minimum size specifications stored in the hint HINT_MINIMUM_SIZE. If HINT_MINIMUM_SIZE is not set, the structure passed to the object will remain empty.

Source: Unrestricted.

Destination: Any generic object.

Parameters: *minSize* A pointer to an empty **GetSizeArgs** structure.

Return: The **GetSizeArgs** structure will contain the object's minimum size.

Interception: Generally not intercepted.

■ MSG_GEN_SET_MINIMUM_SIZE

```
void        MSG_GEN_SET_MINIMUM_SIZE(@stack
            byte   updateMode,
            word   count,
            word   height,             /* SpecHeight */
            word   width);             /* SpecWidth */
```

This message sets the minimum allowable size of an object. An object with a minimum size is not allowed to shrink smaller than the bounds set in HINT_MINIMUM_SIZE. This message alters the minimum size by creating or modifying the hint HINT_MINIMUM_SIZE. Due to the nature of hints, there is no guarantee that the specific UI will implement this message in the expected fashion.

This message will modify the geometry of a currently built object, forcing that object to be unbuilt and then built again. Note that the minimum size of an object has nothing to do with the minimized state of an object (for GenDisplays, the iconification of an object) but merely the minimum allowable size of an object in its normal usable state.

Source: Unrestricted.

Destination: Any generic object.

Parameters:

count	The number of children (or zero, if not applicable).
height	The height of each child.
width	The width of the composite.

Interception: Generally not intercepted.

■ MSG_GEN_GET_MAXIMUM_SIZE

```
void        MSG_GEN_GET_MAXIMUM_SIZE(
            GetSizeArgs  *maxSize);
```

This message retrieves the recipient’s maximum size stored in the hint HINT_MAXIMUM_SIZE. If HINT_MAXIMUM_SIZE is not set, the structure passed will remain empty.

Source: Unrestricted.

Destination: Any generic object.

Parameters:

maxSize	A pointer to an empty **GetSizeArgs** structure.

Return: The **GetSizeArgs** structure will contain the object’s maximum size.

Interception: Generally not intercepted.

◆ Objects

■ MSG_GEN_SET_MAXIMUM_SIZE

```
void        MSG_GEN_SET_MAXIMUM_SIZE(
            word   count,
            word   height,                /* SpecHeight */
            word   width);                /* SpecWidth */
```

This message sets the maximum allowable size of an object. An object with a maximum size is not allowed to grow larger than the bounds set with HINT_MAXIMUM_SIZE. This message alters the maximum size by creating or modifying the hint HINT_MAXIMUM_SIZE. Due to the nature of hints, there is no guarantee that the specific UI will implement this message in the expected fashion.

2.9

This message will modify the geometry of a currently built object, forcing that object to be unbuilt and then built again. Note that the maximum size of an object has nothing to do with the maximized state of an object but merely the maximum allowable size of an object in its normal usable state.

Source: Unrestricted.

Destination: Any generic object.

Parameters: *count* The number of children (or zero, if not applicable).

height The height of each child.

width The width of the composite.

Interception: Generally not intercepted.

■ MSG_GEN_GET_FIXED_SIZE

```
void        MSG_GEN_GET_FIXED_SIZE(
            GetSizeArgs *fixedSize);
```

This message retrieves the size stored in HINT_FIXED_SIZE. If HINT_FIXED_SIZE is not set, the structure passed will remain empty.

Source: Unrestricted.

Destination: Any generic object.

Parameters: *fixedSize* A pointer to an empty **GetSizeArgs** structure.

Return: The **GetSizeArgs** structure will contain the object’s fixed size.

Interception: Generally not intercepted.

■ MSG_GEN_SET_FIXED_SIZE

```
void        MSG_GEN_SET_FIXED_SIZE(
            word   count,
            word   height,                /* SpecHeight */
            word   width);                /* SpecWidth */
```

This message sets the fixed size of an object. An object with a fixed size is forced to remain at the size set with HINT_FIXED_SIZE. This message alters this fixed size by creating or modifying HINT_FIXED_SIZE. Due to the nature of hints, there is no guarantee that the specific UI will implement this message in the expected fashion.

Source: Unrestricted.

Destination: Any generic object.

Parameters:

count	The number of children (or zero, if not applicable).
height	The height of each child.
width	The width of the composite.

Interception: Generally not intercepted.

2.10 Activation Messages

MSG_GEN_APPLY, MSG_GEN_PRE_APPLY, MSG_GEN_POST_APPLY, MSG_GEN_RESET, MSG_GEN_MAKE_APPLYABLE, MSG_GEN_MAKE_NOT_APPLYABLE, MSG_GEN_ACTIVATE, MSG_GEN_ACTIVATE_INTERACTION_DEFAULT

These messages only affect objects which contain properties. These property gadgets (such as GenItemGroups, GenValues, GenTexts, etc.) typically operate in one of two modes: *immediate* or *delayed*.

If an object is operating in immediate mode, any changes in the object's state will cause an immediate change to occur in the application (through the sending of an *apply* message in the object's instance data). For example, if a GenBooleanGroup is operating in immediate mode, every time the user makes a selection within the BooleanGroup, those changes will be applied to the application immediately, as they occur.

If instead an object is operating in delayed mode, any changes in the object's state will not be sent out immediately. They will only be sent out at a later time when the object receives instructions to apply its changes. The UI asks a generic object to apply its changes with MSG_GEN_APPLY.

Typically, each properties gadget operating in delayed mode has a modified state, indicating whether the object has changed since it last applied its changes. Upon receiving MSG_GEN_APPLY, the object checks whether it has been modified; if it has, it will send out its apply message in the object's instance data. Either you or the system may send out MSG_GEN_APPLY to force a properties gadget to apply its changes. Sending out MSG_GEN_APPLY clears any modified states associated with a properties gadget (after those properties are applied, of course).

If MSG_GEN_APPLY is sent to either a dialog GenInteraction or another GenInteraction within that dialog, it will be propagated to all property gadgets within that Interaction. In this manner, you can apply multiple properties within a dialog box all at once. A GIT_PROPERTIES GenInteraction automatically sends out MSG_GEN_APPLY upon receiving MSG_GEN_GUP_INTERACTION_COMMAND with the **InteractionCommand** IC_APPLY. (This is typically through an "Apply" trigger in the dialog box.) The dialog will also send MSG_GEN_PRE_APPLY and MSG_GEN_POST_APPLY messages before and after the apply, respectively.

MSG_GEN_RESET also exhibits this automatic propagation behavior; if this message is sent to a dialog GenInteraction, it will be sent to all property gadgets within that Interaction. However, generic property gadgets provide no default handling for MSG_GEN_RESET. If you wish for an object to reset its properties upon receiving MSG_GEN_RESET, you must subclass the object and handle this message yourself. A GIT_PROPERTIES GenInteraction automatically sends out MSG_GEN_RESET upon receiving MSG_GEN_GUP_INTERACTION_COMMAND with the **InteractionCommand** IC_RESET. (This is typically through a "Reset" trigger in the dialog box.)

MSG_GEN_MAKE_APPLYABLE instructs an object to enable any apply and reset mechanisms that may be disabled. The object must be in delayed mode.

MSG_GEN_MAKE_NOT_APPLYABLE instructs an object to disable any apply and reset mechanisms. The object must be in delayed mode.

MSG_GEN_ACTIVATE activates a generic object as if it had been activated by the user through the specific UI. For example, sending a GenTrigger MSG_GEN_ACTIVATE will cause the trigger to send out its action message in the same manner as if the user had clicked on the trigger.

MSG_GEN_ACTIVATE_INTERACTION_DEFAULT sends a MSG_GEN_ACTIVATE to the object marked as the default within a GenInteraction. This message travels up the generic tree to the first independently-displayable object (usually a dialog box) and then activates the default object for that windowed object.

■ MSG_GEN_APPLY

```
void      MSG_GEN_APPLY();
```

This message is sent to property gadgets to cause them to dispatch their "apply" message and at the same time reset any modified states associated with that object. If this message is sent to a GenInteraction within a dialog box, the GenInteraction will propagate this message to all of its children.

This message may be sent by a GIT_PROPERTIES Interaction upon receipt of MSG_GEN_GUP_INTERACTION_COMMAND with IC_APPLY. You may send this message yourself if you wish for changes in state within the dialog box to be reflected in your application.

Source: Usually a GIT_PROPERTIES Interaction (with an IC_APPLY trigger). You may send this message yourself to force generic objects to apply their changes.

Destination: Any GenInteraction within a dialog box, or any gadget with properties (GenItemGroup, GenValue, etc.).

Parameters: None.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_PRE_APPLY

```
Boolean   MSG_GEN_PRE_APPLY();
```

A properties dialog will send this message before MSG_GEN_APPLY. Geodes may use this to do validation of settings or to prepare for applying of changes.

Source: Normally sent by properties dialog before sending MSG_GEN_APPLY.

Destination: Same as MSG_GEN_APPLY.

Parameters: None.

Return: Return *true* if error.

Interception: Must be intercepted to handle properties validation.

■ MSG_GEN_POST_APPLY

```
void        MSG_GEN_POST_APPLY();
```

A properties dialog will send this message after MSG_GEN_APPLY. This can be used to clean up after application of changes.

Source: Normally sent by properties dialog after sending MSG_GEN_APPLY.

Destination: Same as MSG_GEN_APPLY.

Parameters: None.

Return: Nothing.

Interception: Can be intercepted to handle clean up work after MSG_GEN_APPLY.

■ MSG_GEN_RESET

```
void        MSG_GEN_RESET();
```

This message, if sent to a GenInteraction within a GIT_PROPERTIES dialog box, will propagate to all children below the receiving Interaction. This message provides a convenient means to reset properties within gadgets. There is no default handling of MSG_GEN_RESET by generic objects. You must provide the reset behavior yourself by subclassing individual objects.

Source: Usually a GIT_PROPERTIES GenInteraction (with an IC_RESET trigger), or your application.

Destination: Any GenInteraction within a dialog box, or any gadget subclassed to handle MSG_GEN_RESET.

Parameters: None.

Return: Nothing.

Interception: Your subclassed generic object should intercept this message to provide its own custom reset behavior.

■ MSG_GEN_MAKE_APPLYABLE

```
void        MSG_GEN_MAKE_APPLYABLE();
```

This message is typically sent to a dialog GenInteraction to enable its default “apply” and “reset” buttons. This message also sets up the proper user and actual states for the object. This message is useful in the implementation of custom gadgets.

Source: Unrestricted.

Destination: Any GS_USABLE **GenClass** object. (This message is only meaningful for objects within a GIT_PROPERTIES GenInteraction, however.)

Parameters: None.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_MAKE_NOT_APPLYABLE

```
void        MSG_GEN_MAKE_NOT_APPLYABLE();
```

This message is typically sent to a dialog GenInteraction to disable its default “apply” and “reset” buttons. This message is useful in the implementation of custom gadgets.

Source: Unrestricted.

Destination: Any GS_USABLE **GenClass** object. (This message is only meaningful for objects within a GIT_PROPERTIES GenInteraction, however.)

Parameters: None.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_ACTIVATE

```
void        MSG_GEN_ACTIVATE();
```

This message manually activates an object as if the user had activated it in the manner defined by the specific UI. For example, this message may activate GenTriggers without use of the mouse. This message causes all normal visual cues associated with the activation.

Source: Unrestricted.

Destination: Usually sent by an object to itself.

Parameters: None.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_ACTIVATE_INTERACTION_DEFAULT

```
Boolean  MSG_GEN_ACTIVATE_INTERACTION_DEFAULT();
```

This message acts on a default exclusive object within a window, forcing that object to activate as in MSG_GEN_ACTIVATE. This message travels up the visible tree to the first WIN_GROUP (windowed) object and sends a MSG_GEN_ACTIVATE to the default object selected there. (This default object is typically marked with the hint HINT_DEFAULT_DEFAULT_ACTION.)

Source: Unrestricted.

Destination: Any GS_USABLE **GenClass** object (usually within a dialog box).

Parameters: None.

Return: Will return *true* if the object was activated, *false* if it was not.

Interception: Generally not intercepted.

2.11 Focus Modifications

MSG_GEN_BRING_TO_TOP, MSG_GEN_LOWER_TO_BOTTOM

MetaClass provides the main means of changing the focus and target hierarchies within an application. **GenClass** also provides two messages to perform special cases with windowed objects.

MSG_GEN_BRING_TO_TOP brings a windowed object to the front of all overlapping windowed objects within its window layer. Depending on the specific UI, this message may alter the focus and target by sending MSG_META_GRAB_FOCUS_EXCL or MSG_META_GRAB_TARGET_EXCL to the window brought to the front.

MSG_GEN_LOWER_TO_BOTTOM lowers a windowed object to the bottom of all overlapping windowed objects within its window layer. Depending on the specific UI, this message may alter the focus and target hierarchies by sending a MSG_META_RELEASE_FOCUS_EXCL and MSG_META_RELEASE_TARGET_EXCL to the window lowered to the bottom.

■ MSG_GEN_BRING_TO_TOP

```
void        MSG_GEN_BRING_TO_TOP();
```

This message brings a windowed object to the front if it is in a maximized or overlapping arrangement with other window objects. This message may also grab the focus and target exclusives for the window involved, if the specific UI allows such behavior.

Source: Unrestricted.

Destination: Any GS_USABLE **GenClass** object. (Only GenDisplays, independently displayable GenInteractions, and GenPrimarys are affected.)

Parameters: None.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_LOWER_TO_BOTTOM

```
void        MSG_GEN_LOWER_TO_BOTTOM();
```

This message lowers a windowed object to the bottom if it is in an overlapping arrangement with other windowed objects. This message may also release the focus and target exclusives for the window involved, if the specific UI allows such behavior.

Source: Unrestricted.

Destination: Any GS_USABLE **GenClass** object. (Only GenDisplays, independently displayable GenInteractions, and GenPrimarys are affected.)

Parameters: None.

Return: Nothing.

Interception: Generally not intercepted.

2.12 Navigation Methods

MSG_GEN_NAVIGATE_TO_NEXT_FIELD,
MSG_GEN_NAVIGATE_TO_PREVIOUS_FIELD,
MSG_GEN_NAVIGATION_QUERY

These messages move the navigation cursor between fields at the same level. Typically, these messages alter the focus and target exclusives to implement this behavior. These messages are rarely used within applications.

■ MSG_GEN_NAVIGATE_TO_NEXT_FIELD

```
void        MSG_GEN_NAVIGATE_TO_NEXT_FIELD();
```

This message moves the navigation cursor from its current object to the next valid object. This message is useful if you are trapping keyboard characters using MSG_META_KBD_CHAR and wish to use special characters to navigate the cursor to various fields within a View.

Source: Normally sent by the specific UI, although your application could also send this.

Destination: Any GS_USABLE **GenClass** object.

Parameters: None.

Return: Nothing.

Interception: Generally not intercepted. To alter navigation functionality, intercept MSG_GEN_NAVIGATION_QUERY instead.

■ MSG_GEN_NAVIGATE_TO_PREVIOUS_FIELD

```
void        MSG_GEN_NAVIGATE_TO_PREVIOUS_FIELD();
```

This message moves the navigation cursor from its current object to the previous valid object. This message is useful if you are trapping keyboard characters using MSG_META_KBD_CHAR and wish to use special characters to navigate the cursor to various fields within a View.

Source: Normally sent by the specific UI, although your application could also send this.

Destination: Any GS_USABLE **GenClass** object.

Parameters: None.

Return: Nothing.

Interception: Generally not intercepted. To alter navigation functionality, intercept MSG_GEN_NAVIGATION_QUERY instead.

■ MSG_GEN_NAVIGATION_QUERY

```
void      MSG_GEN_NAVIGATION_QUERY(
          NavigationQueryParams   *retValue,
          optr                    queryOrigin,
          word                    navFlags);
```

2.12

This message can be used by applications to initiate navigation-oriented queries, as the MSG_GEN_NAVIGATE_TO_[NEXT | PREVIOUS]_FIELD messages do.

This structure's return values determine how it fits into the navigation network. If the *NQP_focusableFlag* is set, then the object which should receive the focus was found, and the *NQP_nextObject* field will hold the pointer of this object.

If the *NQP_focusableFlag* is not set, then the focusable object has not yet been found. The *NQP_nextObject* field contains the field of the next object to try, the *NQP_navFlags* contains the updated navigation flags which should be passed in the next query.

Source: Unrestricted.

Destination: Any generic object.

Parameters:

navFlags	Flags giving state of navigation.
queryOrigin	Object which originated this query.
retValue	Structure to hold return values.

Return: Nothing returned explicitly. The *retValue* structure will be filled.

Structures:

```
typedef struct {
      optr        NQP_nextObject;
      word        NQP_navFlags;
      byte        NQP_focusableFlag;
      byte        NQP_unused;
} NavigationQueryParams;
```

◆ Objects

2.13 Window Positions and Sizes

MSG_GEN_SET_WIN_POSITION, MSG_GEN_SET_WIN_SIZE, MSG_GEN_SET_WIN_CONSTRAIN

These messages allow window objects to override their default window behavior. Sending these messages may cause the objects involved to be specifically rebuilt.

2.13

You may also override the default constraints on a windowed object. To do so, send the windowed object MSG_GEN_SET_WIN_CONSTRAIN, passing it the **WinConstrainType** to set for the window.

■ MSG_GEN_SET_WIN_POSITION

```
void    MSG_GEN_SET_WIN_POSITION(
        word                modeAndType,
        SpecWinSizeSpec     xPosSpec,
        SpecWinSizeSpec     yPosSpec);
```

This message sets a window's position. The message must be passed a **WinPositionType** along with two x and y coordinates passed in **SpecWinSizeSpec** structures. The window's upper left corner will be drawn at the supplied coordinates.

Source: Unrestricted.

Destination: Any GS_USABLE GenDisplay, GenPrimary, or independently-displayable GenInteraction.

Parameters:

modeAndType	**VisUpdateMode** and **WinPositionType**. This message expects a **VisUpdateMode** and **WinPositionType** passed in the parameter *modeAndType*. Use the macro GET_MM_AND_TYPE to combine these two values into one word.
xPosSpec	The new x coordinates of the window.
yPosSpec	The new y coordinates of the window.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_SET_WIN_SIZE

```
void        MSG_GEN_SET_WIN_SIZE(
            word                modeAndType,
            SpecWinSizeSpec     xSizeSpec,
            SpecWinsizeSpec     ySizeSpec);
```

This message sets a window's size. The message must be passed a **WinPositionType** along with two x and y coordinates passed in **SpecWinSizeSpec** structures.

Source: Unrestricted.

Destination: Any GS_USABLE GenDisplay, GenPrimary, or independently-displayable GenInteraction.

Parameters: *modeAndType* **VisUpdateMode** and **WinSizeType**. Use the macro GET_MM_AND_TYPE to combine these two values into one word.

xSizeSpec The new x coordinates of the window.

ySizeSpec The new y coordinates of the window.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_SET_WIN_CONSTRAIN

```
void        MSG_GEN_SET_WIN_CONSTRAIN(
            VisUpdateMode       updateMode,
            byte                constrainType);
```

This message changes a window's constrain type to the passed **WindowConstrainType**.

Source: Unrestricted.

Destination: Any GS_USABLE GenPrimary or GenInteraction object.

Parameters: *updateMode* **VisUpdateMode**.

constrainType **WinConstrainType**.

WCT_NONE
WCT_KEEP_PARTIALLY_VISIBLE
WCT_KEEP_VISIBLE
WCT_KEEP_VISIBLE_WITH_MARGIN

Interception: Generally not intercepted.

◆ Objects

GenApplication

3

The GenApplication object is used as the top object in every application geode. It acts as the head of the generic object tree, and it provides all the functionality necessary for launching and shutting down the application. It is a subclass of **GenClass** and therefore inherits all the instance data and messages of that class.

GenApplicationClass has no inherent visual representation. Instead, the main windows of an application are created and managed by one or more GenPrimary objects, which should be placed both as children of the Application object and on the application's GAGCNLT_WINDOWS notification list (if it should appear when the application starts up).

3.1 GenApplication Basics

The top-level generic object of your application must be a GenApplication object. You should place this application object within its own resource; this ensures that your application will take up little memory when minimized. Your application tree should branch from this single node. Code Display 3-1 shows a section of Hello World to illustrate the typical use of a **GenApplicationClass** object.

Code Display 3-1 HelloApp from Hello World

```
/*                      Application Object
 * The hello.gp file contains an "appobj" statement which indicates that this
 * "HelloApp" object is the top-level UI object. Note that the name of the
 * resource you place the application object in may be whatever you choose;
 * it does not have to be AppResource. */

@start AppResource;             /* Begin definition of objects in AppResource. */

@object GenApplicationClass HelloApp = {
    GI_comp = @HelloPrimary;
                        /* The GI_comp attribute lists the generic children
                         * of the object. The HelloApp object has just one
                         * child, the primary window of the application. */
    gcnList(MANUFACTURER_ID_GEOWORKS, GAGCNLT_WINDOWS) = HelloPrimary;
                        /* The window’s application GCN list determines which
                         * windowable children should be launched on
                         * startup. The primary window in most cases should be
                         * launched on startup.*/
}

@end AppResource                /* End definition of objects in AppResource. */
```

Typically, you will not subclass **GenApplicationClass**. You may occasionally send messages to it, but otherwise it exists primarily to interact with the User Interface.

Note that the Application object you set up in your **.goc** file must be reflected in the Geode Parameters file (the **.gp** file). The name of the Application object should appear in the **appobj** field of the **.gp** file.

3.1.1 Instance Data

GenApplicationClass provides several instance data fields, most of which you will not use. All the instance fields of GenApplication are listed in Code Display 3-2 for reference, however.

Code Display 3-2 GenApplication Instance Fields

```
/* These fields will not be used directly. They can be accessed dynamically,
 * however, with the various messages that set and retrieve the instance data. */

@instance AppInstanceReference  GAI_appRef = {““,””,NullHandle,{0}};
@instance word                  GAI_appMode = 0;
@instance AppLaunchFlags        GAI_launchFlags = 0;
@instance byte                  GAI_optFlags = 0;
@instance word                  GAI_appFeatures = 0;
@instance Handle                GAI_specificUI = 0;
@instance ApplicationStates     GAI_states = AS_FOCUSABLE | AS_MODELABLE;
@instance AppAttachFlags        GAI_attachFlags = 0;
@instance UIInterfaceLevel      GAI_appLevel = UIIL_ADVANCED;
@instance ChunkHandle           GAI_iacpConnects = 0;

/* ApplicationStates */
```

```
typedef WordFlags ApplicationStates;
#define AS_HAS_FULL_SCREEN_EXCL         0x2000
#define AS_SINGLE_INSTANCE              0x1000
#define AS_QUIT_DETACHING               0x0800
#define AS_AVOID_TRANSPARENT_DETACH     0x0400
#define AS_TRANSPARENT_DETACHING        0x0200
#define AS_REAL_DETACHING               0x0100
#define AS_QUITTING                     0x0080
#define AS_DETACHING                    0x0040
#define AS_FOCUSABLE                    0x0020
#define AS_MODELABLE                    0x0010
#define AS_NOT_USER_INTERACTABLE        0x0008
#define AS_RECEIVED_APP_OBJECT_DETACH   0x0004
#define AS_ATTACHED_TO_STATE_FILE       0x0002
#define AS_ATTACHING                    0x0001

/* Optimization Flags */

typedef ByteFlags AppOptFlags;
#define AOF_MULTIPLE_INIT_FILE_CATEGORIES0x80

/* GenApplicationClass also modifies two GenClass instance fields. */

@default GI_states = @default & ~GS_USABLE;
@default GI_attrs = @default | GA_TARGETABLE;
```

GAI_appRef is internal. It stores information needed to reload this application. If the application is detached, this instance field contains information necessary to reload this application to its state at detachment.

GAI_appMode stores the message that should be sent to the application's Process object to bring the application back from a saved state. This is initially null and is set by the **GenProcessClass** object as soon as it is determined.

GAI_launchFlags stores the **AppLaunchFlags** that govern how the application should be run. These flags are used internally and are set when the application is first launched.

GAI_appFeatures stores a word representing the application's features as determined by the user's level of expertise. This field is used primarily by hints in GenControl objects and is rarely used directly otherwise.

GAI_specificUI stores the handle of the specific UI under which this application is running. This is determined and set by the system when the application is launched.

GAI_states stores the **ApplicationStates** of the GenApplication. See “ApplicationStates” on page 298 for full information on application states.

GAI_attachFlags stores the **AppAttachFlags** relating to restoring the application from a state file when attached. These flags are only used if the

3.1

Code Display 3-3 GenApplication Vardata Fields

```
@vardata void HINT_APP_IS_ENTERTAINING;
@vardata void HINT_APP_IS_EDUCATIONAL;
@vardata void HINT_APP_IS_PRODUCTIVITY_ORIENTED;

@vardata optr ATTR_GEN_APPLICATION_PRINT_CONTROL;
        @reloc ATTR_GEN_APPLICATION_PRINT_CONTROL, 0, optr;
@vardata optr ATTR_GEN_APPLICATION_KBD_OBJ;
@vardata optr ATTR_GEN_APPLICATION_SAVE_OPTIONS_TRIGGER;
        @reloc ATTR_GEN_APPLICATION_SAVE_OPTIONS_TRIGGER, 0, optr;

/* GenApplication adds a TravelOption to communicate with the Print Control. */

typedef enum {
    TO_PRINT_CONTROL=_FIRST_GenApplicationClass
} GenApplicationTravelOption;
```

GenApplication also provides several hints that indicate the type of application. HINT_APP_IS_ENTERTAINING, HINT_APP_IS_EDUCATIONAL and HINT_APP_IS_PRODUCTIVITY_ORIENTED are provided for this purpose.

ATTR_GEN_APPLICATION_PRINT_CONTROL stores the optr of the object to act as the destination for any messages sent to the **GenApplicationTravelOption** TO_PRINT_CONTROL. Specifically, this attribute is designed to allow remote printing capabilities.

ATTR_GEN_APPLICATION_KBD_OBJ stores the optr of the object to act as the application’s floating keyboard. This object must be a subclass of **GenInteractionClass** and must be in the generic tree below the application object. MSG_GEN_APPLICATION_DISPLAY_FLOATING_KEYBOARD will display this keyboard.

ATTR_GEN_APPLICATION_SAVE_OPTIONS_TRIGGER contains the optr of the Save Options trigger within the options menu. If you have a custom Save Options trigger, you should add the optr of this object in this field.

3.1.2 Application GCN Lists

The GCN mechanism is fully discussed in “General Change Notification,” Chapter 9 of the Concepts Book.

Your application may use application GCN lists to notify objects of certain events. For example, it is essential that any windowed objects that you wish to appear upon startup (including your GenPrimary) are added to the GAGCNLT_WINDOWS GCN list.

GenApplication uses its own GCN list types. The four most often used, and the four you will likely use in most of your applications, are listed below. The others can be found in the file **geoworks.def** and following this short list. Note that all of these GCN list types correspond to the manufacturer ID MANUFACTURER_ID_GEOWORKS.

- MGCNLT_ACTIVE_LIST
  This GCN list keeps a record of all objects that need to be built upon application startup. Objects on this list will receive several system attach and detach messages. Several types of GenControl objects need to be on this list to function properly. In that case, these controllers must also appear on either the GAGCNLT_SELF_LOAD_OPTIONS or GAGCNLT_STARTUP_LOAD_OPTIONS lists as well. Objects that do not need to receive attach notification (but do need to receive detach notification) may be added dynamically to this list.

- GAGCNLT_WINDOWS
  This GCN list keeps a record of windowed objects. All windowed objects that should be visible on startup must be added to this list. Any time a windowed object is visually initialized, it will be added to this list. It will not be removed until the object is visually closed. This is used to save window state across shutdown.

- GAGCNLT_SELF_LOAD_OPTIONS
  This GCN list contains a record of all objects that save options upon receiving MSG_META_SAVE_OPTIONS. These objects will self-load their options; objects on this list will not receive MSG_META_LOAD_OPTIONS

automatically. If they need to do so, they should be added to the GAGCNLT_STARTUP_LOAD_OPTIONS list instead. Objects on this list will be sent MSG_META_SAVE_OPTIONS when the GenApplication receives MSG_META_SAVE_OPTIONS.

- GAGCNLT_STARTUP_LOAD_OPTIONS
This GCN list contains a record of all objects that should both load their options upon startup and save their options upon receiving MSG_META_SAVE_OPTIONS. Objects on this list will receive MSG_META_LOAD_OPTIONS when they are first loaded.

Code Display 3-4 Sample GenApplication with Controllers

```
/* This application includes six controllers. One, the TabControl, must receive
 * MSG_META_ATTACH to work properly and is placed on the GCN active list. Another
 * controller, the GenViewController, must receive MSG_META_LOAD_OPTIONS at
 * startup and is therefore placed on the STARTUP_LOAD_OPTIONS list. All other
 * controllers are placed on the SELF_LOAD_OPTIONS list. Note that controllers
 * placed on the active list still need to be placed on one options list. */

@object GenApplicationClass MyApplication = {
    GI_comp = @MyPrimary;
        /* Windows GCN list. */
    gcnList(MANUFACTURER_ID_GEOWORKS, GAGCNLT_WINDOWS) = @MyPrimary;
        /* Active GCN list. All objects that should receive
         * MSG_META_ATTACH should be on this list. These controllers
         * should also be added to the appropriate LOAD_OPTIONS list. */
    gcnList(MANUFACTURER_ID_GEOWORKS, MGCNLT_ACTIVE_LIST) = @MyTabControl,
                        @MyToolControl;
        /* Startup Load Options GCN list. This list must include
         * all objects that should receive MSG_META_LOAD_OPTIONS
         * at attach time. */
    gcnList(MANUFACTURER_ID_GEOWORKS, GAGCNLT_STARTUP_LOAD_OPTIONS) =
                        @MyGenViewControl;
        /* Self Load Options GCN list. All objects that save
         * options and are not on the Startup Load Options list
         * should appear here. */
    gcnList(MANUFACTURER_ID_GEOWORKS, GAGCNLT_SELF_LOAD_OPTIONS) = @MyTabControl,
                        @MyToolControl, @MyEditControl, @MyCharControl,
                        @MyParaControl;
}
```

Objects

The other GenApplication-defined GCN lists are listed below with comments about their functions. Other GCN list types are also declared by other classes (e.g. **MetaClass**).

GAGCNLT_GEN_CONTROL_NOTIFY_STATUS_CHANGE
Keeps the GenToolControl up-to-date on the status of all the GenControl objects. The data block passed with this list is of type **NotifyGenControlStatusChange**.

GAGCNLT_APP_TARGET_NOTIFY_SELECT_STATE_CHANGE
Notifies objects of changes in the selection state. The data block passed with this list is of type **NotifySelectStateChange**.

GAGCNLT_EDIT_CONTROL_NOTIFY_UNDO_STATE_CHANGE
Notifies objects of changes in the state of the undo item. The data block passed with this list is of type **NotifyUndoStateChange**.

GAGCNLT_APP_TARGET_NOTIFY_TEXT_CHAR_ATTR_CHANGE
Notifies objects of changes in the text character attributes.The data block passed with this list is of type **VisTextNotifyCharAttrChange**.

GAGCNLT_APP_TARGET_NOTIFY_TEXT_PARA_ATTR_CHANGE
Notifies objects of changes in the text paragraph attributes. The data block passed with this list is of type **VisTextNotifyParaAttrChange**.

GAGCNLT_APP_TARGET_NOTIFY_TEXT_TYPE_CHANGE
Notifies objects of changes in the text type change.The data block passed with this list is of type **VisTextNotifyTypeChange**.

GAGCNLT_APP_TARGET_NOTIFY_TEXT_SELECTION_CHANGE
Notifies objects of changes in the text selection. The data block passed with this list is of type **VisTextNotifySelectionChange**.

GAGCNLT_APP_TARGET_NOTIFY_TEXT_COUNT_CHANGE
Notifies objects of changes in the text count. The data block passed with this list is of type **VisTextNotifyCountChange**.

GAGCNLT_APP_TARGET_NOTIFY_STYLE_TEXT_CHANGE
Notifies objects of possible changes in the style. The data block passed with this list is of type **NotifyStyleChange**.

GAGCNLT_APP_TARGET_NOTIFY_STYLE_SHEET_TEXT_CHANGE
Notifies objects of possible changes in the style sheet.The data block passed with this list is of type **NotifyStyleSheetChange**.

GAGCNLT_APP_TARGET_NOTIFY_TEXT_STYLE_CHANGE
Notifies objects of changes in the current text style. The data block passed with this list is of type **NotifyTextStyleChange**.

GAGCNLT_APP_TARGET_NOTIFY_FONT_CHANGE
Notifies objects of changes in the font. The data block passed with this list is of type **NotifyFontChange**.

GAGCNLT_APP_TARGET_NOTIFY_POINT_SIZE_CHANGE
Notifies objects of changes in text point size. The data block passed with this list is of type **NotifyPointSizeChange**.

GAGCNLT_APP_TARGET_NOTIFY_FONT_ATTR_CHANGE
Notifies objects of changes in the font attributes. The data block passed with this list is of type **NotifyFontAttrChange**.

GAGCNLT_APP_TARGET_NOTIFY_JUSTIFICATION_CHANGE
Notifies objects of changes in the paragraph justification. The data block passed with this list is of type **NotifyJustificationChange**.

GAGCNLT_APP_TARGET_NOTIFY_TEXT_FG_COLOR_CHANGE
Notifies objects of changes in the text foreground (character) color. The data block passed with this list is of type **NotifyColorChange**.

GAGCNLT_APP_TARGET_NOTIFY_TEXT_BG_COLOR_CHANGE
Notifies objects of changes in the text background color. The data block passed with this list is of type **NotifyColorChange**.

GAGCNLT_APP_TARGET_NOTIFY_PARA_COLOR_CHANGE
Notifies objects of changes in the text paragraph color.The data block passed with this list is of type **NotifyColorChange**.

GAGCNLT_APP_TARGET_NOTIFY_BORDER_COLOR_CHANGE
Notifies objects of changes in text border color. The data block passed with this list is of type **NotifyColorChange**.

GAGCNLT_APP_TARGET_NOTIFY_SEARCH_SPELL_CHANGE
Notifies objects of changes in the search/spell objects.

GAGCNLT_APP_TARGET_NOTIFY_SEARCH_REPLACE_CHANGE
Notifies objects of changes in the search-and-replace mechanism.

GAGCNLT_APP_TARGET_NOTIFY_CHART_TYPE_CHANGE
Notifies objects of changes in chart type.

GAGCNLT_APP_TARGET_NOTIFY_CHART_GROUP_FLAGS
Notifies objects of changes in chart group flags.

◆Objects

GAGCNLT_APP_TARGET_NOTIFY_CHART_AXIS_ATTRIBUTES
Notifies objects of changes in chart axis attributes.

GAGCNLT_APP_TARGET_NOTIFY_CHART_MARKER_SHAPE
Notifies objects of changes in the chart marker shape.

GAGCNLT_APP_TARGET_NOTIFY_GROBJ_CURRENT_TOOL_CHANGE
Notifies objects of changes in the selected GrObj tool.

GAGCNLT_APP_TARGET_NOTIFY_GROBJ_BODY_SELECTION_STATE_CHANGE
Notifies objects of changes in the GrObj body selection state.

GAGCNLT_APP_TARGET_NOTIFY_GROBJ_AREA_ATTR_CHANGE
Notifies objects of changes in the GrObj area attributes.

GAGCNLT_APP_TARGET_NOTIFY_GROBJ_LINE_ATTR_CHANGE
Notifies objects of changes in the GrObj line attributes.

GAGCNLT_APP_TARGET_NOTIFY_GROBJ_TEXT_ATTR_CHANGE
Notifies objects of changes in the GrObj text attributes.

GAGCNLT_APP_TARGET_NOTIFY_STYLE_GROBJ_CHANGE
Notifies objects of changes in style changes relating to the GrObj.

GAGCNLT_APP_TARGET_NOTIFY_STYLE_SHEET_GROBJ_CHANGE
Notifies objects of changes in the style sheet.

GAGCNLT_APP_TARGET_NOTIFY_RULER_TYPE_CHANGE
Notifies objects of changes in ruler type.

GAGCNLT_APP_TARGET_NOTIFY_RULER_GRID_CHANGE
Notifies objects of changes in the ruler grid.

GAGCNLT_TEXT_RULER_OBJECTS
Notifies objects of changes in the active ruler.

GAGCNLT_APP_TARGET_NOTIFY_BITMAP_CURRENT_TOOL_CHANGE
Notifies objects of changes in the selected bitmap tool.

GAGCNLT_APP_TARGET_NOTIFY_BITMAP_CURRENT_FORMAT_CHANGE
Notifies objects of changes in the current bitmap format.

GAGCNLT_APP_TARGET_NOTIFY_FLAT_FILE_FIELD_PROPERTIES_STATUS_CHANGE
Notifies objects of changes in the flatfile database properties status.

GAGCNLT_APP_TARGET_NOTIFY_FLAT_FILE_FIELD_LIST_CHANGE
Notifies objects of changes in the flatfile field list.

GAGCNLT_APP_TARGET_NOTIFY_FLAT_FILE_RCP_STATUS_CHANGE
Notifies objects of changes in flatfile status.

GAGCNLT_APP_TARGET_NOTIFY_FLAT_FILE_FIELD_APPEARANCE_CHANGE
Notifies objects that a field within the flat file has changed its appearance.

GAGCNLT_APP_NOTIFY_DOC_SIZE_CHANGE
Notifies objects of changes in document size. The data block passed with this list is of type **NotifyPageSetupChange**.

GAGCNLT_APP_NOTIFY_PAPER_SIZE_CHANGE
Notifies objects of changes in chosen paper size. The data block passed with this list is of type **NotifyPageSetupChange**.

GAGCNLT_APP_TARGET_NOTIFY_VIEW_STATE_CHANGE
Notifies objects of changes in of view state. The data block passed with this list is of type **NotifyViewStateChange**.

GAGCNLT_CONTROLLED_GEN_VIEW_OBJECTS
A list of GenView objects controlled by the GenViewControl. (These GenViews will have ATTR_GEN_VIEW_INTERACT_WITH_CONTROLLER set in their instance data.)

GAGCNLT_APP_TARGET_NOTIFY_INK_STATE_CHANGE
Notifies objects of changes in Ink state.

GAGCNLT_CONTROLLED_INK_OBJECTS
A list of Ink objects controlled by the InkControl.

GAGCNLT_APP_TARGET_NOTIFY_PAGE_STATE_CHANGE
Notifies objects of changes in page state. The data block passed with this list is of type **NotifyPageStateChange**.

GAGCNLT_APP_TARGET_NOTIFY_DOCUMENT_CHANGE
Notifies objects of changes in a document. The data block passed with this list is of type **NotifyPageStateChange**.

GAGCNLT_APP_TARGET_NOTIFY_DISPLAY_CHANGE
Notifies objects of changes in a display. The data block passed with this list is of type **NotifyDisplayChange**.

GAGCNLT_APP_TARGET_NOTIFY_DISPLAY_LIST_CHANGE
Notifies objects of changes in the display list. The data block passed with this list is of type **NotifyColorChange**.

◆Objects

GAGCNLT_APP_TARGET_NOTIFY_SPLINE_MARKER_SHAPE
Notifies objects of changes in a spline marker shape.

GAGCNLT_APP_TARGET_NOTIFY_SPLINE_POINT
Notifies objects of changes in spline points.

GAGCNLT_APP_TARGET_NOTIFY_SPLINE_POLYLINE
Notifies objects of changes in spline polylines.

GAGCNLT_APP_TARGET_NOTIFY_SPLINE_SMOOTHNESS
Notifies objects of changes in spline smoothness.

GAGCNLT_APP_TARGET_NOTIFY_SPLINE_OPEN_CLOSE_CHANGE
Notifies objects of changes in a spline's open/close state.

GAGCNLT_APP_TARGET_NOTIFY_SPREADSHEET_ACTIVE_CELL_CHANGE
Notifies objects of changes in the spreadsheet's active cell range. The data block passed with this list is of type **NotifySSheetActiveCellChanged**.

GAGCNLT_APP_TARGET_NOTIFY_SPREADSHEET_EDIT_BAR_CHANGE
Notifies objects of changes in the spreadsheet's edit bar. The data block passed with this list is of type **NotifySSheetEditBarChanged**.

GAGCNLT_APP_TARGET_NOTIFY_SPREADSHEET_SELECTION_CHANGE
Notifies objects of changes in the spreadsheet's selection. The data block passed with this list is of type **NotifySSheetSelectionChanged**.

GAGCNLT_APP_TARGET_NOTIFY_SPREADSHEET_CELL_WIDTH_HEIGHT_CHANGE
Notifies objects of changes in the spreadsheet's cell width or height. The data block passed with this list is of type **NotifySSheetCellWidthHeightChang**e.

GAGCNLT_APP_TARGET_NOTIFY_SPREADSHEET_DOC_ATTR_CHANGE
Notifies objects of changes in the spreadsheet's document attributes. The data block passed with this list is of type **NotifySSheetDocAttrChange**.

GAGCNLT_APP_TARGET_NOTIFY_SPREADSHEET_CELL_ATTR_CHANGE
Notifies objects of changes in the spreadsheet's cell attributes. The data block passed with this list is of type **NotifySSheetCellAttrChange**.

GAGCNLT_APP_TARGET_NOTIFY_SPREADSHEET_CELL_NOTES_CHANGE
Notifies objects of changes in the notes of a cell within a spreadsheet.

GAGCNLT_APP_TARGET_NOTIFY_SPREADSHEET_DATA_RANGE_CHANGE
Notifies objects of changes in the spreadsheet's data range selection. The

data block passed with this list is of type **NotifySSheetDataRangeChange**.

GAGCNLT_APP_TARGET_NOTIFY_TEXT_NAME_CHANGE
Notifies objects of changes in a text object's names run. The data block passed with this list is of type **VisTextNotifyNameChange.**

GAGCNLT_FLOAT_FORMAT_CHANGE
Notifies objects of changes in a particular float format within the float format controller.

GAGCNLT_DISPLAY_OBJECTS_WITH_RULERS
A list of GenDisplay objects that have rulers.

GAGCNLT_APP_TARGET_NOTIFY_APP_CHANGE
Notifies objects of changes in an application.

GAGCNLT_APP_TARGET_NOTIFY_LIBRARY_CHANGE
Notifies objects of changes in a library.

GAGCNLT_APP_TARGET_NOTIFY_CARD_BACK_CHANGE
Notifies objects that a card back has changed.

GAGCNLT_NOTIFY_FOCUS_TEXT_OBJECT
Notifies objects that a an editable text object has a gained the focus. This list is used by the floating keyboard to determine when it should be enabled or not.

GAGCNLT_NOTIFY_TEXT_CONTEXT
Notifies objects that a the selection or data in a focused text object has changed, if that text object has text contexts turned on. This list is used by hand-writing recognition.

GAGCNLT_NOTIFY_HELP_CONTEXT_CHANGE
Notifies objects that a help context has changed.

GAGCNLT_FLOAT_FORMAT_INIT
Notifies the float controller that it should re-initialize itself. This normally sent to the controller when the target document has changed.

GAGCNLT_ALWAYS_INTERACTABLE_WINDOWS
This list stores windows that should always remain interactable, even if modal windows are on-screen. Objects on this list will get messages even if GenInteractions invoked by **UserDoDialog()** are on-screen. These objects also receive MSG_META_CHECK_IF_INTERACTABLE_OBJECT to

allow them to specify objects under them (such as objects in the child blocks) that should also receive messages.

GAGCNLT_USER_DO_DIALOGS
This list stores all dialog boxes initiated via **UserDoDialog()**.

GAGCNLT_MODAL_WIN_CHANGE
Notifies objects that modal window changes have occurred within the application.

GACGNLT_APP_TARGET_NOTIFY_SPREADSHEET_NAME_CHANGE
Notifies objects that a spreadsheet's name has changed.

GAGCNLT_CONTROLLERS_WITHIN_USER_DO_DIALOGS
This list stores objects (usually controllers) that will appear within the context of **UserDoDialog()** but will not be within the same block as the dialog box. Objects on this list will receive MSG_META_CHECK_IF_INTERACTABLE_OBJECT.

GAGCNLT_FOCUS_WINDOW_KBD_STATUS
Notifies floating keyboards when windows gain the focus on pen systems.

3.1.3 Application Instance Reference

```
GAI_appRef, GAI_appMode, GAI_optFlags,
MSG_GEN_APPLICATION_SET_APP_MODE_MESSAGE,
MSG_GEN_APPLICATION_GET_APP_MODE_MESSAGE,
MSG_GEN_APPLICATION_SET_APP_INSTANCE_REFERENCE,
MSG_GEN_APPLICATION_GET_APP_INSTANCE_REFERENCE,
MSG_GEN_APPLICATION_SEND_APP_INSTANCE_REFERENCE_TO_FIELD
```

GAI_appRef stores information (within an **AppInstanceReference** structure) that allows a GenApplication object to be reloaded from its former state. This structure contains a path name, long file name, and disk handle of a state file as well as an additional byte of disk data. The system automatically manages this state file and this instance field.

GAI_appMode stores the application message that should be sent to the process to bring this application back from a saved state. This is initially null (unless previously saved to a state file); it is set by **GenProcessClass** when a mode is determined at MSG_META_ATTACH time. You should not alter this instance field.

GAI_optFlags stores miscellaneous optimization flags. The only flag at this time—AOF_MULTIPLE_INIT_FILE_CATEGORIES—indicates that within this application there may be several different init file categories (marked with ATTR_GEN_INIT_FILE_CATEGORY). This allows MSG_META_GET_INI_CATEGORY to perform a full upward recursive search to find the appropriate init file category; by default, if an init file category is not found on an object, only the GenApplication object is queried.

3.1

■ MSG_GEN_APPLICATION_SET_APP_MODE_MESSAGE

```
void      MSG_GEN_APPLICATION_SET_APP_MODE_MESSAGE(
          Message modeMessage);
```

This message stores a message into the GenApplication's *GAI_appMode* field. Generally, this message indicates the current mode of the application. Should the application be shut down and restored, this message will be sent to the Process object to restore the state to the same mode.

Source: Rarely used.

Destination: Any GenApplication object.

Parameters: *modeMessage* The message number to be stored in *GAI_appMode*.

Return: Nothing.

Interception: Do not intercept.

■ MSG_GEN_APPLICATION_GET_APP_MODE_MESSAGE

```
Message   MSG_GEN_APPLICATION_GET_APP_MODE_MESSAGE();
```

This message returns the message number stored in the GenApplication's *GAI_appMode* field.

Source: Rarely used.

Destination: Any GenApplication object.

Parameters: None.

Return: The message number stored in *GAI_appMode*.

Interception: Do not intercept.

■ MSG_GEN_APPLICATION_SET_APP_INSTANCE_REFERENCE

```
void      MSG_GEN_APPLICATION_SET_APP_INSTANCE_REFERENCE(
          Handle appInstance);
```

This message sets the *GAI_appRef* field to the passed structure.

Source: Rarely used.

Destination: Any GenApplication object.

Parameters: *appInstance* The handle of the reference data block.

Return: Nothing.

Interception: Do not intercept.

■ MSG_GEN_APPLICATION_GET_APP_INSTANCE_REFERENCE

```
Handle MSG_GEN_APPLICATION_GET_APP_INSTANCE_REFERENCE();
```

This message retrieves the values in *GAI_appRef*.

Source: Rarely used.

Destination: Any GenApplication object.

Parameters: None.

Return: The handle of the reference stored in *GAI_appRef*.

Interception: Do not intercept.

■ MSG_GEN_APPLICATION_SEND_APP_INSTANCE_REFERENCE_TO_FIELD

```
void MSG_GEN_APPLICATION_SEND_APP_INSTANCE_REFERENCE_TO_FIELD();
```

This message causes the GenApplication to send the contents of its *GAI_appRef* field off to its parent GenField object.

Source: Rarely used.

Destination: Any GenApplication object.

Interception: Do not intercept.

3.1.4 Attach and Launch Flags

```
GAI_launchFlags, GAI_attachFlags,
MSG_GEN_APPLICATION_GET_LAUNCH_FLAGS,
MSG_GEN_APPLICATION_GET_ATTACH_FLAGS
```

GAI_launchFlags stores flags that are passed when the application is first launched. These flags are never set within your object declaration but may be passed with other messages.

ALF_SEND_LAUNCH_REQUEST_TO_UI_TO_HANDLE

If this bit is set, the application will not immediately be launched but will instead wait for the UI to launch it. This flag should not be set by the application itself; it is only used by **UserLoadApplication()**.

ALF_OPEN_IN_BACK

If this bit is set, the application will be opened behind other applications, in an inactive state. This flag is only used with MSG_GEN_PROCESS_OPEN_APPLICATION.

ALF_DESK_ACCESSORY

If this bit is set, the application will be treated as a "desk accessory," in a layer above normal applications.

ALF_DO_NOT_OPEN_ON_TOP

If this bit is set, the application will be prevented from both gaining the focus and opening on top of other applications.

ALF_OVERRIDE_MULTIPLE_INSTANCE

If this bit is set, and UILM_MULTIPLE_INSTANCES is also set as the application's **UILaunchModel**, the application will not query the user if he or she attempts to initiate multiple instances of the same application; the application will be multiply launched without first checking whether the already running application should be used instead.

ALF_OPEN_FOR_IACP_ONLY

If this bit is set, the application will be opened only to facilitate a connection with **IACPConnect()**; the application should close once that task is completed. If the application should remain open after such an IACP connection, this bit should be cleared. This behavior is used only for MSG_GEN_PROCESS_OPEN_APPLICATION connections. The application cannot be opened in engine mode.

GAI_attachFlags stores flags related to an application attaching from a state file. These flags are never set within your object declaration but may be passed with other messages.

AAF_RESTORING_FROM_STATE

If this bit is set, the application was launched via MSG_GEN_PROCESS_RESTORE_FROM_STATE. AAF_STATE_FILE_PASSED will also be set.

AAF_STATE_FILE_PASSED
If this bit is set, the application is being restored from a state file.

AAF_DATA_FILE_PASSED
If this bit is set, a data file containing much of the instance data of the application (of type **AppLaunchBlock**) is being passed to the launching message. This is internal and should not be used.

AAF_RESTORING_FROM_QUIT
If this bit is set, the application was in the process of quitting, reached engine mode, and is now being started up into application mode again before completely exiting. If set, then AAF_RESTORING_FROM_STATE will also be set. This bit ensures that certain clean-up operations that are done before an application is quit are un-done, and that the application returns to its former state.

■ MSG_GEN_APPLICATION_GET_LAUNCH_FLAGS

```
AppLaunchFlags MSG_GEN_APPLICATION_GET_LAUNCH_FLAGS();
```

This message retrieves the contents of the GenApplication's *GAI_launchFlags* field.

Source: Rarely used.

Destination: Any GenApplication object.

Parameters: None.

Return: The **AppLaunchFlags** record stored in *GAI_launchFlags*.

Interception: Do not intercept.

■ MSG_GEN_APPLICATION_GET_ATTACH_FLAGS

```
AppAttachFlags MSG_GEN_APPLICATION_GET_ATTACH_FLAGS();
```

This message retrieves the contents of the GenApplication's *GAI_attachFlags* field.

Source: Rarely used.

Destination: Any GenApplication object.

Parameters: None.

Return: The **AppAttachFlags** stored in *GAI_attachFlags*.

Interception: Do not intercept.

3.1.5 ApplicationStates

```
GAI_states, MSG_GEN_APPLICATION_GET_STATE,
MSG_GEN_APPLICATION_SET_STATE,
MSG_GEN_APPLICATION_SET_NOT_USER_INTERACTABLE,
MSG_GEN_APPLICATION_SET_USER_INTERACTABLE,
MSG_GEN_APPLICATION_SET_NOT_QUITTING,
MSG_GEN_APPLICATION_SET_ATTACHED_TO_STATE_FILE,
MSG_GEN_APPLICATION_SET_NOT_ATTACHED_TO_STATE_FILE
```

GAI_states stores the **ApplicationStates** of the application. By default, a GenApplication is both AS_FOCUSABLE and AS_MODELABLE, therefore enabling those hierarchies for this application. Only under extremely rare conditions will you alter this behavior. The flags of **ApplicationStates** are listed below:

AS_HAS_FULL_SCREEN_EXCL
This bit is set if the application is currently the top screen object at its level. This bit may only be set if the application is between receiving a MSG_META_GAINED_FULL_SCREEN_EXCL and a MSG_META_LOST_FULL_SCREEN_EXCL.

AS_SINGLE_INSTANCE
This bit is set if the application is not capable of being launched multiple times. You should not need to set this unless

AS_QUIT_DETACHING
If this bit is set, the detach sequence has been initiated as the result of a QUIT. This bit will only be set if AS_QUITTING is also set; the bit is set in the UI thread at the same time MSG_META_DETACH is sent to the process. Therefore, this bit represents an intermediate step between AS_QUITTING and AS_DETACHING.

AS_AVOID_TRANSPARENT_DETACH
This bit is set if the application should not be transparently detached. If the application is running within

UILM_TRANSPARENT mode, then the application will not detach when another application is launched.

AS_TRANSPARENT_DETACHING
This bit is set if the application is being transparently detached. An application can be transparently detached if another application is started in this application's field and that field is marked UILM_TRANSPARENT.

AS_REAL_DETACHING
This bit is set if MSG_GEN_PROCESS_REAL_DETACH has been sent to the process, signalling the irreversible demise of the application. This bit is only set if the UI has finished detaching and the GS_USABLE bit on the application has been cleared.

AS_QUITTING
The application is currently quitting.

AS_DETACHING
The application object has received MSG_META_DETACH and is detaching.

AS_FOCUSABLE
The application may receive the focus exclusive from its field parent. When launched, if this flag is set, the application automatically grabs the focus. This flag is set by default.

AS_MODELABLE
The application may receive the model exclusive from its field parent. When launched, if this flag is set, the application will automatically grab the model exclusive. This flag is set by default.

AS_NOT_USER_INTERACTABLE
The application should not be interactable with the user. This prevents the user from navigating to non-visible applications or otherwise selecting the application.

AS_RECEIVED_APP_OBJECT_DETACH
The application has received a detach message.

AS_ATTACHED_TO_STATE_FILE
The application is currently attached to a state file.

AS_ATTACHING
The application is currently attaching (processing MSG_META_ATTACH).

MSG_GEN_APPLICATION_GET_STATE

```
ApplicationStates MSG_GEN_APPLICATION_GET_STATE();
```

This message retrieves the current application state, stored in *GAI_states*.

Source: Rarely used.

Destination: Any GenApplication object.

Parameters: None.

Return: The **ApplicationStates** record stored in *GAI_states*.

Interception: Do not intercept.

MSG_GEN_APPLICATION_SET_STATE

```
void        MSG_GEN_APPLICATION_SET_STATE(
            ApplicationStates      set,
            ApplicationStates      clear);
```

This message alters a GenApplication's *GAI_states* flags. This message should only be used to set flags that aren't set internally by the UI. Flags that can be altered are the AS_FOCUSABLE, AS_MODELABLE, AS_NOT_USER_DETACHABLE and AS_AVOID_TRANSPARENT_DETACH state bits.

This message does not reject attempts to set internal bits; therefore, be careful in using this message and only use it to set the external bits mentioned above.

Source: Unrestricted. This message is also used internally.

Destination: Any GenApplication object.

Parameters: *set* **ApplicationStates** to set.

clear **ApplicationStates** to clear.

Return: Nothing.

Warnings: Do not attempt to set any internal **ApplicationStates** bits with this message.

Interception: May intercept, but must pass to superclass at some point.

■ MSG_GEN_APPLICATION_SET_NOT_QUITTING

```
void        MSG_GEN_APPLICATION_SET_NOT_QUITTING();
```

This message clears the AS_QUITTING bit in the application's *GAI_states* bitfield.

Source: Sent by the UI or the kernel.

Destination: A GenApplication object.

Interception: Do not intercept.

■ MSG_GEN_APPLICATION_SET_NOT_USER_INTERACTABLE

```
void        MSG_GEN_APPLICATION_SET_NOT_USER_INTERACTABLE();
```

This message sets the AS_NOT_USER_INTERACTABLE flag in the application's *GAI_states* field.

Source: Infrequently used.

Destination: The GenApplication to be made not interactable.

Interception: Do not intercept.

■ MSG_GEN_APPLICATION_SET_USER_INTERACTABLE

```
void        MSG_GEN_APPLICATION_SET_USER_INTERACTABLE();
```

This message clears the AS_NOT_USER_INTERACTABLE flag in the application's *GAI_states* field.

Source: Infrequently used.

Destination: The GenApplication to be made interactable.

Interception: Do not intercept.

■ MSG_GEN_APPLICATION_SET_ATTACHED_TO_STATE_FILE

```
void        MSG_GEN_APPLICATION_SET_ATTACHED_TO_STATE_FILE();
```

This message sets the AS_ATTACHED_TO_STATE_FILE in the GenApplication's *GAI_states* field.

Source: Sent by the UI or the kernel.

Destination: The GenApplication object that has been attached to a state file.

Interception: Do not intercept.

■ **MSG_GEN_APPLICATION_SET_NOT_ATTACHED_TO_STATE_FILE**

```
void        MSG_GEN_APPLICATION_SET_NOT_ATTACHED_TO_STATE_FILE();
```

This message clears the AS_ATTACHED_TO_STATE_FILE in the GenApplication's *GAI_states* field.

Source: Sent by the UI or the kernel.

Destination: The GenApplication object that has been detached to a state file.

Interception: Do not intercept.

3.1.6 Application Features and Levels

```
GAI_appFeatures, GAI_appLevel,
MSG_GEN_APPLICATION_GET_APP_FEATURES,
MSG_GEN_APPLICATION_SET_APP_FEATURES,
MSG_GEN_APPLICATION_SET_APP_LEVEL,
MSG_GEN_APPLICATION_UPDATE_APP_FEATURES,
MSG_GEN_APPLICATION_UPDATE_FEATURES_VIA_TABLE,
GenAppUsabilityTupleFlags, GenAppUsabilityTuple,
GenAppUsabilityCommands
```

A GenApplication may store a word of features (*GAI_appFeatures*); these features correspond to groups of UI objects. Depending on a certain feature being set or not set, certain groups of UI objects may or may not appear, allowing you to customize your application for different users or criteria. It is up to your application to define both the features and the objects that these features correspond to.

GAI_appFeatures is a word-length bitfield. Each bit corresponds to a certain group of features which you define. You may thus have up to 16 different feature groups for any application. (Note that each feature group may include several UI objects.) In general, you group these features together so that they correspond to a specific **UIInterfaceLevel**. If the application appears at a different User Interface level, the makeup of the UI will be different.

An application's user interface level is stored in the GenApplication's *GAI_appLevel* instance data entry. Each **UIInterfaceLevel** corresponds to a certain group of features. Changing the UI level changes the group of features that may be displayed.

The features represented in the bitfield may be represented in hints added to GenControl objects. Most often, the controllers and the application will adjust their menus, tools, and other UI gadgetry to conform to the features specified in this record.

Code Display 3-5 Setting Up Features

```
/* Features are stored in a word-length bitfield. */

typedef WordFlags MyFeatures;

@define MF_EDIT_FEATURES        (0x8000)
@define MF_PASTE_FEATURES       (0x4000)
@define MF_FORMAT_FEATURES      (0x2000)

/* We might want to group certain features together based on the level of
 * expertise of the user. In this example, if the user level is “intermediate”
 * (which we will define later), we allow features for editing and pasting to the
 * UI. If the user level is “advanced” we allow the intermediate features and also
 * allow formatting features. */

@define INTRODUCTORY_FEATURES   (0)

@define INTERMEDIATE_FEATURES   (@MF_EDIT_FEATURES | @MF_PASTE_FEATURES)

@define ADVANCED_FEATURES       (@INTERMEDIATE_FEATURES | @MF_FORMAT_FEATURES)
```

MSG_GEN_APPLICATION_GET_APP_FEATURES returns the current application features and **UIInterfaceLevel** in use by an application.

You may set the application’s *GAI_appFeatures* by sending it MSG_GEN_APPLICATION_SET_APP_FEATURES. You may also change the application’s user level by sending it MSG_GEN_APPLICATION_SET_APP_LEVEL. Each of these messages in turn generates a MSG_GEN_APPLICATION_UPDATE_APP_FEATURES.

This message is meant to be sub-classed so that you can alter the behavior for different features. In most cases, however, you will simply handle this message, fill in relevant parameters, and send the GenApplication MSG_GEN_APPLICATION_UPDATE_FEATURES_VIA_TABLE—the message which performs the actual work done in changing the UI. This message expects a number of arguments.

The most important argument is the table of **GenAppUsabilityTuple** entries that correspond to each feature. These entries define what sort of UI change is required, and what object is required to change.You must set up this table beforehand.

The types of usability commands available (in the bit positions set aside with GAUTF_COMMAND in the tuple's **GenAppUsabilityTupleFlags**) are:

- GAUC_USABILITY
  If set, the object should be made GS_USABLE if the feature is on. This is the default behavior.

- GAUC_RECALC_CONTROLLER
  If set, the controller needs to have its features recalculated. The particular feature bits are stored with the controller itself; the application knows nothing more about the controller's features other than that they need to be changed.

- GAUC_REPARENT
  If set, the object needs to be relocated to another part of the UI, underneath a different parent. This parent is passed in the *reparentObject* entry for MSG_GEN_APPLICATION_UPDATE_FEATURES_VIA_TABLE. Only one object may be re-parented for each application.

- GAUC_POPUP
  If set, the object should be made a popup menu. Note that this allows a sub-group to become a menu without having to use GAUC_REPARENT.

- GAUC_TOOLBAR
  If set, the object is a GenBoolean that corresponds to a toolbar state. Turning the feature on or off forces the GenBoolean to send an apply in addition to other behavior.

- GAUC_RESTART
  If set, the object needs to be "kick started" by first setting it GS_NOT_USABLE and then GS_USABLE.

Because each feature may have multiple objects affected, the **GenAppUsabilityTupleFlags** entry GAUTF_END_OF_LIST indicates that there are no more commands for that feature. The flag GAUTF_OFF_IF_BIT_IS_ON indicates that a given command should be reversed for the object. (I.e. if the feature is on, the object should be removed, not added.)

Code Display 3-6 Setting Up the GenAppUsabilityTuple Tables

```
/* Each GenAppUsabilityTuple will refer to a specific set of features. */

/*
 * Since GUAC_USABILITY is the default setting (and is zero) setting any other
 * flags either supersedes or complements this behavior. In this case, setting the
 * EditToolEntry as a GUAC_TOOLBAR command supersedes the GUAC_USABILITY command.
 * Setting the GUATF_END_OF_LIST flag for the EditTrigger does not alter the
 * GUAC_USABILITY command, which is still implicit.
 */

static const GenAppUsabilityTuple editFeaturesList [] =
{
        {GUAC_TOOLBAR,          @EditToolEntry  },
        {GUATF_END_OF_LIST      @EditTrigger    }
};

static const GenAppUsabilityTuple pasteFeaturesList [] =
{
        {GUAC_END_OF_LIST,      @PasteTrigger   }
};

static const GenAppUsabilityTuple formatFeaturesList [] =
{
        {
        GAUTF_END_OF_LIST | GUAC_RECALC_CONTROLLER,
        @FeatureController
        }
};

/* After each feature’s GenAppUsabilityTuple is set up, you should set up a table
 * of these structures to pass to relevant messages. */

static const GenAppUsabilityTuple * const usabilityTable [] =
{
        editFeaturesList,
        pasteFeaturesList,
        formatFeaturesList
};
```

```
/*
 * Within your code, decide where you wish to set the application features
 * (usually within some sort of User level dialog box that passes a selection of
 * feature bits) and send either MSG_GEN_APPLICATION_SET_APP_FEATURES or
 * MSG_GEN_APPLICATION_SET_APP_LEVEL with the proper feature bits set.
 */

@method MyLevelApplicationClass, MSG_MY_APPLICATION_SET_USER_LEVEL
{
   @call oself::MSG_GEN_APPLICATION_SET_APP_FEATURES(selection);
}

/*
 * Then intercept MSG_GEN_APPLICATION_UPDATE_APP_FEATURES and set up the correct
 * parameters for MSG_GEN_APPLICATION_UPDATE_FEATURES_VIA_TABLE.
 */

@method MyLevelApplicationClass, MSG_GEN_APPLICATION_UPDATE_APP_FEATURES
{
    @call oself::MSG_GEN_APPLICATION_UPDATE_FEATURES_VIA_TABLE(
        NullOptr,
        @ObjectToReparent,       /* if any */
        levelTable,              /* if any */
        sizeof(usabilityTable) / sizeof(usabilityTable [0]),
        usabilityTable,
        appOpeningFlag,
        oldLevel,
        level,
        featuresChanged,
        featuresOn);
}
```

■ MSG_GEN_APPLICATION_GET_APP_FEATURES

dword MSG_GEN_APPLICATION_GET_APP_FEATURES();

This message retrieves the set of features set for the application.

Source: Unrestricted—typically a GenControl object finding out the application's UI level.

Destination: The GenApplication running the controller.

Parameters: None.

Return: A dword containing the word of features stored in *GAI_appFeatures* and the **UIInterfaceLevel** for the application. The features are stored in the high word; the interface level is stored in the low word.

Interception: Generally not intercepted.

■ MSG_GEN_APPLICATION_SET_APP_FEATURES

```
void    MSG_GEN_APPLICATION_SET_APP_FEATURES(
        word   features);
```

This message sets a new set of features into the GenApplication's *GAI_appFeatures* record. This message in turn generates a MSG_GEN_APPLICATION_UPDATE_APP_FEATURES for your application object to intercept. (The message handler for that message must in turn send MSG_GEN_APPLICATION_UPDATE_FEATURES_VIA_TABLE to activate the UI changes specified in the features list.)

Source: Unrestricted—typically a system function.

Destination: The GenApplication having its features set.

Parameters: *features* The new word-sized record of application features to set.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_APPLICATION_UPDATE_APP_FEATURES

```
void    MSG_GEN_APPLICATION_UPDATE_APP_FEATURES(@stack
        optr                unReparentObject,
        optr                reparentObject,
        GenAppUsabilityTuple*levelTable,
        word                tableLength,
        void                *table,
        word                appOpeningFlag,
        UIInterfaceLevel    oldLevel,
        UIInterfaceLevel    level,
        word                featuresChanged,
        word                featuresOn);
```

This message is sent by the application to itself when it is told to change either its features or its **UIInterfacelevel**. This message is passed a number of parameters, most of which should simply be passed to MSG_GEN_APPLICATION_UPDATE_FEATURES_VIA_TABLE. If you have reparent objects (or un-reparent objects), you must set them up here.

Source: Sent by an application object to itself in response to a MSG_GEN_APPLICATION_SET_APP_FEATURES or MSG_GEN_APPLICATION_SET_APP_LEVEL.

Destination: The GenApplication object.

Parameters:

unReparentObject The optr of the object to be unreparented. If a null optr is passed, the object will by default be moved up and added as the next sibling of its current parent.

reparentObject The optr of the object to be reparented to another UI location. You must supply this object if you have a **GenAppUsabilityTuple** entry that contains a GAUC_REPARENT entry.

levelTable This table contains the GenAppUsabilityTuples corresponding to objects that contain their *own* features and that must be notified when the user level changes. This is so that those objects can generate their own feature updates. Typically, controllers are included among these objects.

tableLength The number of table entries in *table*.

table Table of **GenAppUsabilityTuple** entries that must be updated when the user level changes. This table is usually set up as global data and maps each user level feature to a **GenAppUsabilityTuple**.

appOpeningFlag Set if the application is starting.

oldLevel The previous **UIInterfaceLevel**.

level The new **UIInterfaceLevel**.

featuresChanged The set of features changed (deleted).

featuresOn The set of features to set on.

Interception: To set an application’s features, you must intercept this message and send MSG_GEN_APPLICATION_UPDATE_FEATURES_VIA_TABLE; there is no default message handler. This message is provided as a convenient point to intercept and change features before executing the changes.

Objects

■ MSG_GEN_APPLICATION_UPDATE_FEATURES_VIA_TABLE

```
void    MSG_GEN_APPLICATION_UPDATE_FEATURES_VIA_TABLE(
        optr                unReparentObject,
        optr                reparentObject,
        GenAppUsabilityTuple*levelTable,
        word                tableLength,
        void                *table,
        word                appOpeningFlag,
        UIInterfaceLevel    oldLevel,
        UIInterfaceLevel    level,
        word                featuresChanged,
        word                featuresOn);
```

This message is called to update the application's features to reflect a new set of features to

Source: Typically, your message handler for MSG_GEN_APPLICATION_UPDATE_APP_FEATURES.

Destination: The GenApplication object.

Parameters:

unReparentObject The optr of the object to be unreparented. If a null optr is passed, the object will by default be moved up and added as the next sibling of its current parent.

reparentObject The optr of the object to be reparented to another UI location. You must supply this object if you have a **GenAppUsabilityTuple** entry that contains a GAUC_REPARENT entry.

levelTable This table contains the GenAppUsabilityTuples corresponding to objects that contain their *own* features and that must be notified when the user level changes. This is so that those objects can generate their own feature updates. Typically, controllers are included among these objects.

tableLength The number of table entries in *table*.

table Table of **GenAppUsabilityTuple** entries that must be updated when the user level changes. This table is usually set up as global data and maps each user level feature to a **GenAppUsabilityTuple**.

appOpeningFlag Set if the application is starting.

oldLevel	The previous **UIInterfaceLevel**.
level	The new **UIInterfaceLevel**.
featuresChanged	The set of features changed (deleted).
featuresOn	The set of features to set on.

Interception: Generally not intercepted. Intercept MSG_GEN_APPLICATION_UPDATE_APP_FEATURES instead.

3.1

3.1.7 IACP

`GAI_iacpConnects, MSG_GEN_APPLICATION_IACP_REGISTER, MSG_GEN_APPLICATION_IACP_UNREGISTER, MSG_GEN_APPLICATION_IACP_GET_NUMBER_OF_CONNECTIONS, MSG_GEN_APPLICATION_IACP_ GET_NUMBER_OF_APP_MODE_CONNECTIONS, MSG_GEN_APPLICATION_IACP_SHUTDOWN_ALL_CONNECTIONS, MSG_GEN_APPLICATION_IACP_COMPLETE_CONNECTIONS, MSG_GEN_APPLICATION_APP_MODE_COMPLETE`

IACP (the GEOS *I*nter *A*pplication *C*ommunication *P*rotocol) allows applications to communicate with each other. IACP is flexible enough to let applications know whether another application is open, closed, or in the process of attaching or detaching. IACP allows applications to convey information to one another, and could be used to support updating data (e.g. documents) across applications. The IACP mechanism is discussed more fully in “Applications and Geodes,” Chapter 6 of the Concepts Book. The information included below only discusses **GenApplicationClass** support of IACP mechanisms.

GAI_iacpConnects stores the chunk handle to an array of active IACP connections. This chunk stores the IACP connection value referring to the remote application and the type of IACP connection (i.e. a connection that is enabled during a MSG_GEN_PROCESS_OPEN_APPLICATION, for example). These values are manipulated internally and there is no need to access them. You will instead use a variety of messages provided with **GenApplicationClass** to register and unregister for application notification.

A good deal of support has been added to **GenApplicationClass** to support IACP. The main things you need to know about this support are:

- A GenApplication object will refuse to quit so long as there are IACP connections open to it. It can, however, be forcibly detached, as happens when the system shuts down. In such a case, it will call **IACPShutdownAll()** to shut down all remaining connections either to or from it.
- When it receives MSG_META_IACP_LOST_CONNECTION sent to it as a server, it will eventually call **IACPShutdown()** for the connection when it is certain no more messages relating to the connection are in any relevant queue. It will forward this message to all GenDocument objects below any GenDocumentGroup object in the application, so they can close themselves if the lost connection was the last reference to them.
- It will automatically register itself as a server for the application's token, either when it receives MSG_META_APP_STARTUP if the **AppLaunchBlock** indicates it's running in engine mode, or when it receives MSG_META_ATTACH and has attached all the various pieces of UI. It will unregister itself as a server when it loses its last IACP connection and is no longer functioning in application mode (either because the user quit the application long since, or because it was never functioning in application mode).
- It registers and unregisters itself by sending MSG_GEN_APPLICATION_IACP_REGISTER and MSG_GEN_APPLICATION_IACP_UNREGISTER to itself, allowing an application to subclass these messages and register with other lists as appropriate.
- To determine whether it has any IACP connections remaining, it invokes MSG_GEN_APPLICATION_IACP_GET_NUMBER_OF_CONNECTIONS on itself. Should an application register the GenApplication or any other object as a server for another list, it can determine how many connections remain from that list and augment the number returned by the default handler in **GenApplicationClass**. If this returns non-zero, GenApplication will not shut down the application voluntarily.
- After unregistering, it will force-queue a message to itself that will check the number of connections again. If the number has become non-zero between the unregister and the check, it re-registers and does not shut down. If the number remains zero, however, the application will exit.

- When forcibly detached, it will send MSG_GEN_APPLICATION_IACP_SHUTDOWN_ALL_CONNECTIONS to itself. The default handler will call **IACPShutdownAll()**, passing its own optr. A subclass can use this to perform a similar operation for any other server objects the application might have.

■ MSG_GEN_APPLICATION_IACP_REGISTER

```
void        MSG_GEN_APPLICATION_IACP_REGISTER();
```

3.1

This message is sent by a GenApplication object to itself when it registers for IACP. It is not a message meant to be sent externally to an application to register it for IACP. Instead, you can subclass this message and register the object with other lists.

Source: Sent by the GenApplication object to itself.

Destination: The GenApplication object to register with IACP.

Interception: May be intercepted if there are other lists with which you want to register the application, or other server objects. You must make sure to call the superclass, however.

■ MSG_GEN_APPLICATION_IACP_UNREGISTER

```
void        MSG_GEN_APPLICATION_IACP_UNREGISTER();
```

This message is sent by a GenApplication object to itself when it unregisters for IACP. It is not a message meant to be sent externally to an application to unregister it for IACP. Instead, you can subclass this message and unregister the object with other lists.

Source: Sent by the GenApplication object to itself.

Destination: The GenApplication object to unregister with IACP.

Interception: May be intercepted if there are other lists with which you need to unregister the application, or other server objects. You must make sure to call the superclass, however.

■ MSG_GEN_APPLICATION_IACP_GET_NUMBER_OF_CONNECTIONS

```
word        MSG_GEN_APPLICATION_IACP_GET_NUMBER_OF_CONNECTIONS();
```

This message returns the number of active engine or app-mode IACP connections for a given application. This message is used to check whether an application open only for IACP purposes may be closed.

◆ Objects

Source: Called by GenProcessClass when the UI has finished processing MSG_META_QUIT for the application; this determines if the application should exit at this point or if there are client applications that need the application to stay open.

Destination: GenApplication object of the application.

Return: Number of open connections. If non-zero, application will remain open.

Interception: Only intercept if you have other server objects beside your GenApplication object. If intercepting, call the superclass first and then add the number of connections to the other objects onto the result returned by **GenApplicationClass**.

■ MSG_GEN_APPLICATION_IACP_GET_NUMBER_OF_APP_MODE_CONNECTIONS

```
void        MSG_GEN_APPLICATION_IACP_GET_NUMBER_OF_APP_MODE_CONNECTIONS();
```

This message retrieves the number of connections which require that the application be open in app-mode (as opposed to engine mode). This message is used to check whether an application can be closed down into engine mode even if some IACP connections are still open.

Source: This message is called by the **GenProcessClass** when the UI has finished processing MSG_META_QUIT to determine if the application should really close down to engine mode.

Destination: GenApplication object of application.

Interception: Only intercept if you have other server objects besides your GenApplication object. You should call the superclass first and then add the number of connections to other objects onto the result returned by **GenApplicationClass**.

■ MSG_GEN_APPLICATION_IACP_SHUTDOWN_ALL_CONNECTIONS

```
void        MSG_GEN_APPLICATION_IACP_SHUTDOWN_ALL_CONNECTIONS();
```

This message shuts down all IACP connections for a given application, either on the server or the client side of the connection.

Source: Sent by the GenApplication object to itself.

Destination: GenApplication object of the application.

Interception: May be intercepted to allow connections to other server objects to be shut down. You must call the superclass at some point to ensure that application connections are shut down as well.

■ MSG_GEN_APPLICATION_APP_MODE_COMPLETE

```
void        MSG_GEN_APPLICATION_APP_MODE_COMPLETE();
```

This message is sent to the application when its life as a user-interactable app is complete. The default behavior is to continue shutting down the process if there are no IACP connections active.

Source: Sent by the GenProcess object after it receives a MSG_META_ACK from detaching the application.

Destination: GenApplication object.

Interception: Generally not intercepted; If you have other server connections which you want taken into account before shutting the application completely down, you should intercept MSG_GEN_APPLICATION_IACP_GET_NUMBER_OF_CONNECTIONS instead.

■ MSG_GEN_APPLICATION_IACP_COMPLETE_CONNECTIONS

```
void        MSG_GEN_APPLICATION_IACP_COMPLETE_CONNECTIONS();
```

This message completes all pending IACP connections, accepting any queued messages that have been waiting to be handled. If you subclass it, be sure to call the superclass at some point.

Source: Sent by the GenApplication object to itself in its default MSG_GEN_APPLICATION_OPEN_COMPLETE method, as we assume that the object should be able to handle IACP messages at this latter stage of opening the IACP mechanism.

Destination: GenApplication object of the application.

Interception: May be intercepted if there are other lists besides those connected to the application's token, if those other connections might be pending. If intercepting, you must call the superclass eventually.

3.2 Advanced GenApplication Usage

Typically, you will merely set up a GenApplication object in your **.goc** file and then leave it alone. You may occasionally send it messages to invoke functions or to query the application. These messages are infrequently used, however, and you will likely not have need for them.

3.2.1 An Application's Life Cycle

For information on how an application is launched and closed, see "Applications and Geodes," Chapter 6 of the Concepts Book.

3.2.2 Application Busy States

```
MSG_GEN_APPLICATION_MARK_BUSY,
MSG_GEN_APPLICATION_MARK_NOT_BUSY,
MSG_GEN_APPLICATION_HOLD_UP_INPUT,
MSG_GEN_APPLICATION_RESUME_INPUT,
MSG_GEN_APPLICATION_IGNORE_INPUT,
MSG_GEN_APPLICATION_ACCEPT_INPUT,
MSG_GEN_APPLICATION_MARK_APP_COMPLETELY_BUSY,
MSG_GEN_APPLICATION_MARK_APP_NOT_COMPLETELY_BUSY
```

An application's busy state is reflected by its cursor. An application may have several busy states, set appropriately for the action going on at the time. The messages below can set the application's busy state. You will not usually send any of these messages to your GenApplication object. Instead, you will usually set appropriate *GI_attrs* to automatically send out these messages during times when the application will be busy.

MSG_GEN_APPLICATION_MARK_BUSY marks the application busy (usually by changing the cursor to an appropriate shape determined by the specific UI) until the current operation in the application thread completes. This message is sent by UI gadgets that have GA_INITIATES_BUSY_STATE set in their *GI_attrs* fields. It may also be called by any other object that wants to mark the application busy. When an application is busy, the user may continue to interact with it.

MSG_GEN_APPLICATION_MARK_NOT_BUSY removes the busy state marker. This message is automatically sent to the application object when the operation that initiated the busy state completes.

3.2 MSG_GEN_APPLICATION_HOLD_UP_INPUT instructs the User Interface to place all input events into a special “hold-up” queue until the input is resumed. This message also marks the application busy. Applications marked GA_INITIATES_INPUT_HOLD_UP will receive this message whenever they initiate an operation.

MSG_GEN_APPLICATION_RESUME_INPUT removes the input hold-up state, allowing normal input flow. This message flushes the “hold-up” event queue into the application’s input queue, ensuring that all events during the “hold up” operation are handled before any new events.

MSG_GEN_APPLICATION_IGNORE_INPUT instructs the GenApplication object to ignore all input events it receives. This may be accompanied with an audible warning (beep). Applications marked GA_INITIATES_INPUT_IGNORE will receive this message whenever they initiate an operation.

MSG_GEN_APPLICATION_ACCEPT_INPUT removes the input ignore state and directs the GenApplication object to again receive input events and handle them.

All of these messages are cumulative. The application will keep track of how many times each of these messages is sent. For example, each MSG_GEN_APPLICATION_MARK_NOT_BUSY message will remove a MSG_GEN_APPLICATION_MARK_BUSY. When the count reaches zero, the busy state is removed.

■ MSG_GEN_APPLICATION_MARK_BUSY

```
void      MSG_GEN_APPLICATION_MARK_BUSY();
```

This message marks the application busy and changes the cursor image.

Source: Sent automatically by objects with GA_INITIATES_BUSY_STATE set.

Destination: The GenApplication object running the sender.

◆ Objects

Interception: Do not intercept.

■ MSG_GEN_APPLICATION_MARK_NOT_BUSY

```
void    MSG_GEN_APPLICATION_MARK_NOT_BUSY();
```

This message marks the application not busy, removing the effect of a previous MSG_GEN_APPLICATION_MARK_BUSY.

Source: Sent automatically by objects with GA_INITIATES_BUSY_STATE set.

Destination: The GenApplication object running the sender.

Interception: Do not intercept.

■ MSG_GEN_APPLICATION_HOLD_UP_INPUT

```
void    MSG_GEN_APPLICATION_HOLD_UP_INPUT();
```

This message causes the GenApplication to mark itself busy and redirect input events to a special “hold-up” queue. When the application is ready to resume normal activity, it first handles the messages in the hold-up queue before handling new input messages.

Source: Used infrequently.

Destination: The GenApplication object to be held up.

Interception: Do not intercept.

■ MSG_GEN_APPLICATION_RESUME_INPUT

```
void    MSG_GEN_APPLICATION_RESUME_INPUT();
```

This message causes a GenApplication to resume normal input handling after it has been held up with MSG_GEN_APPLICATION_HOLD_UP_INPUT.

Source: Used infrequently.

Destination: The GenApplication object running the sender.

Interception: Do not intercept.

■ MSG_GEN_APPLICATION_IGNORE_INPUT

```
void    MSG_GEN_APPLICATION_IGNORE_INPUT();
```

This message causes the GenApplication to consume all input events it receives rather than handle them. This message may be used during debugging as a *last* resort to help find synchronization problems.

Source: Infrequently used.

Destination: The GenApplication object to consume input events.

Interception: Do not intercept.

■ MSG_GEN_APPLICATION_ACCEPT_INPUT

```
void        MSG_GEN_APPLICATION_ACCEPT_INPUT();
```

This message undoes a previous MSG_GEN_APPLICATION_IGNORE_INPUT, allowing the GenApplication to once again handle input events normally.

Source: Infrequently used.

Destination: The GenApplication object to resume input handling.

Interception: Do not intercept.

■ MSG_GEN_APPLICATION_MARK_APP_COMPLETELY_BUSY

```
void        MSG_GEN_APPLICATION_MARK_APP_COMPLETELY_BUSY();
```

This message is rarely used and forces a busy state over the application regardless of other states. It should be used only when a time-intensive task is going on in the UI and the program can not handle input during that time.

Source: Infrequently used.

Destination: The GenApplication object to be marked completely busy.

Interception: Do not intercept.

■ MSG_GEN_APPLICATION_MARK_APP_NOT_COMPLETELY_BUSY

```
void        MSG_GEN_APPLICATION_MARK_APP_NOT_COMPLETELY_BUSY();
```

This message undoes MSG_GEN_APPLICATION_MARK_COMPLETELY_BUSY, allowing the application to once again handle user input.

Source: Infrequently used.

Destination: The GenApplication object to be marked not busy.

Interception: Do not intercept.

3.2.3 The GenApplication’s Moniker

MSG_GEN_APPLICATION_FIND_MONIKER,
MSG_GEN_APPLICATION_SET_TASK_ENTRY_MONIKER

Every GenApplication object should be given a moniker; this moniker is displayed by the UI in its task list. (In OSF/Motif, the task list is manifested as the floating “Express” menu.) While you will set the moniker just like for any generic object, the GenApplication has two messages that can be used to get or set the moniker used in the task list.

■ MSG_GEN_APPLICATION_FIND_MONIKER

```
optr      MSG_GEN_APPLICATION_FIND_MONIKER(
          MemHandle           destBlock,
          word                searchFlags,
          DisplayType         displayType);
```

This message finds the specified moniker in the GenApplication’s **VisMonikerList** and optionally copies it into a specified destination block.

Source: Infrequently used.

Destination: The GenApplication to get the moniker from.

Parameters: *destBlock* The handle of the destination block into which the chunk will be copied. For this to work, you must pass VMSF_COPY_CHUNK in *searchFlags*.

searchFlags A record of **VisMonikerSearchFlags** indicating what type of moniker to find in the moniker list and what to do with it when it is found.

displayType The display type of the moniker to search for.

Return: The optr of the GenApplication’s **VisMonikerList** chunk.

Interception: Do not intercept.

■ MSG_GEN_APPLICATION_SET_TASK_ENTRY_MONIKER

```
void      MSG_GEN_APPLICATION_SET_TASK_ENTRY_MONIKER(
          optr   entryMoniker);
```

This message changes the moniker which is used in the GenField’s task list. The task list menu will be updated if currently visible.

Source: Infrequently used.

Destination: The GenApplication to have its moniker changed.

Parameters: *entryMoniker* The optr of the chunk containing the moniker that will be set into the task list.

Return: Nothing.

Interception: Do not intercept.

3.2.4 Measurement Type

MSG_GEN_APPLICATION_SET_MEASUREMENT_TYPE, MSG_GEN_APPLICATION_GET_MEASUREMENT_TYPE, GET_APP_MEASUREMENT_TYPE

Each application has a “measurement type” associated with it. The measurement type indicates whether measurements should default to metric or standard US measurements.

■ MSG_GEN_APPLICATION_SET_MEASUREMENT_TYPE

```
void    MSG_GEN_APPLICATION_SET_MEASUREMENT_TYPE(
        byte   measurementType);
```

This message sets the measurement type used by the application.

Source: Infrequently used.

Destination: The GenApplication to have the new measurement type.

Parameters: *measurementType* A value of **AppMeasurementType** to set for the application. Can be AMT_US, AMT_METRIC, or AMT_DEFAULT.

Return: Nothing.

Interception: Do not intercept.

■ MSG_GEN_APPLICATION_GET_MEASUREMENT_TYPE

```
word    MSG_GEN_APPLICATION_GET_MEASUREMENT_TYPE();
```

This message gets the measurement currently used by the application.

Source: Infrequently used.

Destination: The GenApplication whose measurement is to be retrieved.

Parameters: None.

Return: A word value, the low byte of which represents the application's measurement type. Use the macro GET_MEASUREMENT_TYPE to extract the measurement type from the return value.

Interception: Do not intercept.

3.2.5 Interaction with the UI

The GenApplication is an application's main point of contact with the UI. As such, it has several messages that are sent by the UI or by other objects to initiate certain UI-related functions. These messages will rarely, if ever, be used by application programmers, but they are documented here in case you find them useful.

3.2.5.1 Attaching and Detaching

```
MSG_GEN_APPLICATION_INITIATE_UI_QUIT,
MSG_GEN_APPLICATION_INSTALL_TOKEN,
MSG_GEN_APPLICATION_DETACH_PENDING,
MSG_GEN_APPLICATION_OPEN_COMPLETE,
MSG_GEN_APPLICATION_QUIT_AFTER_UI
```

■ MSG_GEN_APPLICATION_INITIATE_UI_QUIT

```
void    MSG_GEN_APPLICATION_INITIATE_UI_QUIT();
```

This message causes the GenApplication to begin quitting. The application will automatically go through the entire quit sequence.

Source: Infrequently used.

Destination: The GenApplication of the application to be shut down.

Interception: Do not intercept.

■ MSG_GEN_APPLICATION_INSTALL_TOKEN

```
void    MSG_GEN_APPLICATION_INSTALL_TOKEN();
```

This message instructs the GenApplication object to set its token into the token database file.

Source: Infrequently used.

Destination: The GenApplication to have its token installed.

Interception: Do not intercept.

■ MSG_GEN_APPLICATION_DETACH_PENDING

```
void	MSG_GEN_APPLICATION_DETACH_PENDING();
```

This message is sent to the specific UI library through the GenApplication to notify it that the application is about to be shut down. It is used to abort any application-modal dialog boxes so the application's Process object will be able to detach.

Source: The GenApplication object before it detaches.

Destination: The GenApplication of the application about to be detached.

Interception: Do not intercept.

■ MSG_GEN_APPLICATION_OPEN_COMPLETE

```
void	MSG_GEN_APPLICATION_OPEN_COMPLETE();
```

This message is sent by the GenApplication object to itself when it has finished opening (after it has set itself usable). It is sent via the queue and indicates that the application's UI is fully usable.

Source: A GenApplication after it has set itself GS_USABLE.

Destination: Sent to itself.

Interception: Do not intercept.

■ MSG_GEN_APPLICATION_QUIT_AFTER_UI

```
void	MSG_GEN_APPLICATION_QUIT_AFTER_UI();
```

This message is called from the MSG_META_QUIT handler in GenProcessClass, after the UI has finished its MSG_GEN_APPLICATION_INITIATE_UI_QUIT sequence. This message is the application's last chance to abort a quit before the DETACH sequence begins. The default behavior is to abort the QUIT if the application is still open for the user (i.e. not ALF_OPEN_FOR_IACP_CONNECTION_ONLY) or if an IACP connection remains that requires the application to remain open.

Source: **GenProcessClass**.

Destination: GenApplication object of the application.

Interception: May be intercepted and not sent to the superclass to abort the QUIT.

3.2.5.2 Queries of the UI

```
MSG_GEN_APPLICATION_GET_DISPLAY_SCHEME,
MSG_GEN_APPLICATION_QUERY_UI,
MSG_GEN_APPLICATION_NOTIFY_MODAL_WIN_CHANGE,
MSG_GEN_APPLICATION_INK_QUERY_REPLY,
MSG_GEN_APPLICATION_GET_GCN_LIST_OF_LISTS,
MSG_GEN_APPLICATION_TEST_WIN_INTERACTABILITY,
MSG_GEN_APPLICATION_VISIBILITY_NOTIFICATION,
MSG_GEN_APPLICATION_GET_MODAL_WIN,
MSG_GEN_APPLICATION_CHECK_IF_ALWAYS_INTERACTABLE_OBJECT
```

■ MSG_GEN_APPLICATION_GET_DISPLAY_SCHEME

```
void    MSG_GEN_APPLICATION_GET_DISPLAY_SCHEME(
        DisplayScheme * displayScheme);
```

This message gets the current display scheme used by the application.

Source: Infrequently used.

Destination: Any GenApplication.

Parameters: *displayScheme* A pointer to a structure of type **DisplayScheme**. This structure will be filled by the method and contains information about the color scheme, display type, font ID, and point size used by the application.

Return: The **DisplayScheme** structure pointed to by *displayScheme* will be filled upon return.

Interception: Do not intercept.

■ MSG_GEN_APPLICATION_QUERY_UI

```
Handle  MSG_GEN_APPLICATION_QUERY_UI();
```

This message is used to determine which UI should be used at a given point in the generic tree for a certain type of object.

Source: Infrequently used.

Destination: Any GenApplication object.

Parameters: None.

Return: The handle of the specific UI library geode.

Interception: Do not intercept.

■ MSG_GEN_APPLICATION_NOTIFY_MODAL_WIN_CHANGE

```
void    MSG_GEN_APPLICATION_NOTIFY_MODAL_WIN_CHANGE();
```

This message is called on a GenApplication object by the UI whenever the application should check to see if there is a change in modal status. The behavior is to look for the top system-modal window owned by the application and then the top application-modal window within the application's layer.

This message sets the AS_ATTACHED_TO_STATE_FILE in the GenApplication's *GAI_states* field.

Source: Sent by the UI.

Destination: Any GenApplication object.

Interception: Do not intercept.

■ MSG_GEN_APPLICATION_INK_QUERY_REPLY

```
void    MSG_GEN_APPLICATION_INK_QUERY_REPLY(
        InkReturnValue      inkReturnValue,
        word                inkGstate);
```

This message is sent to an Application object in reply to a MSG_META_QUERY_IF_PRESS_IS_INK. It indicates whether the object that was queried can or can not handle Ink presses. The GenApplication object responds by sending a message to the UI.

Source: Sent by an object in response to MSG_META_QUERY_IF_PRESS_IS_INK.

Destination: The GenApplication object associated with the sender.

Parameters: *inkReturnValue* A value indicating whether the object queried can handle Ink input or not. Will be one of IRV_NO_INK, IRV_INK_WITH_STANDARD_OVERRIDE, IRV_DESIRES_INK, or IRV_WAIT.

inkGstate The GState, if any, to be used when drawing Ink.

Return: Nothing.

Interception: Do not intercept.

■ MSG_GEN_APPLICATION_GET_GCN_LIST_OF_LISTS

```
ChunkHandle MSG_GEN_APPLICATION_GET_GCN_LIST_OF_LISTS();
```

This message retrieves the GenApplication's GCN list of lists chunk handle. This chunk handle may then be used with a number of kernel routines for GCN list management or to perform operations on individual GCN lists.

Source: Any object in the GenApplication's thread.

Destination: Any GenApplication object.

Parameters: None.

Return: The chunk handle of the GCN list of lists chunk; a null chunk handle will be returned if the chunk does not exist.

Interception: Do not intercept.

■ MSG_GEN_APPLICATION_TEST_WIN_INTERACTABILITY

```
Boolean   MSG_GEN_APPLICATION_TEST_WIN_INTERACTABILITY(
          optr   inputOD,
          Handle window);
```

This message tests whether the passed window object is interactable.

Source: Unrestricted.

Destination: Any GenApplication object.

Parameters:

inputOD	The optr of the windowed object to be tested.
window	The window handle of the window to be tested.

Return: The return value will be *false* if there are no modal windows in the system or if the window object passed is the topmost active modal window. The return value will be *true* if an active modal window exists and is not the passed window object (in this case, if the passed object has any window grabs, it should release them). *True* is also returned if there is no modal window but the GenApplication is ignoring input.

Interception: Do not intercept.

■ MSG_GEN_APPLICATION_VISIBILITY_NOTIFICATION

```
void      MSG_GEN_APPLICATION_VISIBILITY_NOTIFICATION(
          optr                obj,
          Boolean             opening);
```

Notifies the GenApplication object that it has become visible or not visible.

Source: The specific UI.

Destination: The GenApplication that has become visible or not visible.

Parameters: *obj* The optr of the object sending the notification message.

opening A Boolean indicating the state of the object: *true* if open, *false* if closed.

Return: Nothing.

Interception: Do not intercept.

■ MSG_GEN_APPLICATION_GET_MODAL_WIN

```
optr      MSG_GEN_APPLICATION_GET_MODAL_WIN();
```

This message returns the current top modal window for the application, if any is present.

Source: Unrestricted.

Destination: GenApplication object.

Return: optr of top modal windowed object.

■ MSG_GEN_APPLICATION_CHECK_IF_ALWAYS_INTERACTABLE_OBJECT

```
Boolean   MSG_GEN_APPLICATION_CHECK_IF_ALWAYS_INTERACTABLE_OBJECT(
          optr                objToCheck);
```

This message checks if the passed object should always remain interactable.

Source: Unrestricted.

Destination: GenApplication object.

Parameters: *objToCheck* optr of object to check the interactable state.

Return: *true* if the object is always interactable, *false* if not.

3.2.5.3 Alterations of Functionality

MSG_GEN_APPLICATION_BRING_WINDOW_TO_TOP,
MSG_GEN_APPLICATION_LOWER_WINDOW_TO_BOTTOM,
MSG_GEN_APPLICATION_BUILD_STANDARD_DIALOG,
MSG_GEN_APPLICATION_DO_STANDARD_DIALOG,
MSG_GEN_APPLICATION_TOGGLE_CURSOR,
MSG_GEN_APPLICATION_BRING_UP_HELP,

```
MSG_GEN_APPLICATION_TOGGLE_CURRENT_MENU_BAR,
MSG_GEN_APPLICATION_TOGGLE_FLOATING_KEYBOARD,
MSG_GEN_APPLICATION_TOGGLE_EXPRESS_MENU
```

■ MSG_GEN_APPLICATION_BRING_WINDOW_TO_TOP

```
void      MSG_GEN_APPLICATION_BRING_WINDOW_TO_TOP(
          optr   window);
```

This message brings the passed window to the front of the screen.

Source: Unrestricted

Destination: Any GenApplication object.

Parameters: *window* The optr of the window object to be brought to the front of the screen.

Return: Nothing.

Interception: Do not intercept.

■ MSG_GEN_APPLICATION_LOWER_WINDOW_TO_BOTTOM

```
void      MSG_GEN_APPLICATION_LOWER_WINDOW_TO_BOTTOM(
          optr   window);
```

This message sends the specified window to the back of the screen, behind other window objects.

Source: Unrestricted.

Destination: Any GenApplication object.

Parameters: *window* The optr of the window object to be sent to the back of the screen.

Return: Nothing.

Interception: Do not intercept.

■ MSG_GEN_APPLICATION_BUILD_STANDARD_DIALOG

```
optr      MSG_GEN_APPLICATION_BUILD_STANDARD_DIALOG(
          char   * customTriggers,
          char   * arg2,
          char   * arg1,
          char   * string,
          CustomDialogBoxFlags dialogFlags);
```

This message builds a standard dialog box for the application.

Source: Infrequently used.

Destination: The GenApplication to use the dialog box.

Parameters: *customTriggers* A pointer to a table of custom GenTrigger information. Each trigger given in the table will appear in the dialog box in the order declared. The table is made up of structures of type **StandardDialogResponseTriggerTable**.

*arg*1 A pointer to a character string to be displayed in the dialog box.

*arg*2 A pointer to a second string to be displayed in the dialog box.

string A pointer to a custom character string to be displayed in the dialog box.

dialogFlags A record of **CustomDialogBoxFlags** indicating what type of dialog box is to be created.

Return: The optr of the dialog box object.

Interception: Do not intercept.

■ MSG_GEN_APPLICATION_DO_STANDARD_DIALOG

```
void      MSG_GEN_APPLICATION_DO_STANDARD_DIALOG(@stack
          word   dialogMethod,
          optr   dialogOD,
          char   *helpContext,
          char   * customTriggers,
          char   * arg2,
          char   * arg1,
          char   * string,
          CustomDialogBoxFlags dialogFlags);
```

This message executes a standard dialog box and returns immediately. When the dialog box is shut down, the message passed in the *dialogMethod* parameter is sent to the object specified in *dialogOD*. Only one dialog box at a time may be displayed with this message.

Source: Infrequently used.

Destination: The GenApplication to use the created dialog box.

Parameters: *dialogMethod* The message to be sent out when the user is finished with the dialog. This message should be defined based on the prototype GEN_APP_DO_STANDARD_DIALOG_MSG.

dialogOD The recipient of the message specified in *dialogMethod* above.

helpContext The help context for this dialog box.

customTriggers A pointer to a table of custom GenTrigger information. Each trigger given in the table will appear in the dialog box in the order declared. The table is made up of structures of type **StandardDialogResponseTriggerTable**.

arg2 A pointer to a second string to be displayed in the dialog box.

arg1 A pointer to a character string to be displayed in the dialog box.

string A pointer to a custom character string to be displayed in the dialog box.

dialogFlags A record of **CustomDialogBoxFlags** indicating what type of dialog box is to be created.

Return: Nothing.

Interception: Do not intercept.

■ MSG_GEN_APPLICATION_TOGGLE_CURSOR

```
void        MSG_GEN_APPLICATION_TOGGLE_CURSOR();
```

This message toggles the cursor for a text object.

Source: Infrequently used.

Destination: The GenApplication of the text object.

Interception: Do not intercept.

■ MSG_GEN_APPLICATION_BRING_UP_HELP

```
void        MSG_GEN_APPLICATION_BRING_UP_HELP();
```

This message brings up help for an application. Normally, this is accomplished by sending a message to the focus object telling it to bring up a help window with the focus' help context.

Source: Unrestricted, though generally from an application Help icon or <F1>.

Destination: GenApplication object.

Interception: Generally not intercepted, though it may be useful if for some reason you do not wish to bring up help (such as it doesn't exist for this application).

■ MSG_GEN_APPLICATION_TOGGLE_CURRENT_MENU_BAR

```
void        MSG_GEN_APPLICATION_TOGGLE_CURRENT_MENU_BAR();
```

This message toggles the GIV_POPOUT state of the current GenPrimary's menu bar. This message only takes effect if the menu bar is toggleable (i.e. if UIWO_POPOUT_MENU_BAR is set).

Source: Unrestricted.

Destination: GenApplication object.

■ MSG_GEN_APPLICATION_TOGGLE_FLOATING_KEYBOARD

```
void        MSG_GEN_APPLICATION_TOGGLE_FLOATING_KEYBOARD();
```

This message toggles the state of the floating keyboard within the current application. Applications may subclass this to bring up their own PenInputControl (or equivalent) object. Otherwise the application object will create its own.

Source: Unrestricted, though generally only supported on pen-based systems.

Destination: GenApplication object.

Interception: May be intercepted if the application has its own PenInputControl object.

■ MSG_GEN_APPLICATION_TOGGLE_EXPRESS_MENU

```
void        MSG_GEN_APPLICATION_TOGGLE_EXPRESS_MENU();
```

This message toggles (opens or closes) the parent field's express menu.

Source: Unrestricted.

Destination: GenApplication object.

GenDisplay / GenPrimary

4

Applications communicate with users through a “user interface.” The system needs a way of grouping the user-interface components together. For this reason, most applications will have a GenPrimary object. This object serves as the top-level object in the user interface tree.

GenPrimaryClass is a subclass of **GenDisplayClass**. The GenDisplay object, like the GenPrimary object, manages other pieces of the user interface (menus, triggers, text objects, etc.). Furthermore, the GenDisplay object is a great boon to applications that need to perform several tasks at once (as, for example, a word processor which can have several files open at once). An application can have several GenDisplay objects, all of them children of a GenDisplayGroup object. Collectively, displays and primaries are called *windows*. All windows have certain functionality in common; that functionality can vary according to the specific UI.

4.1

The Document Control objects can be set to create GenDisplay objects when files are opened and to attach them automatically to a GenDisplayGroup. This enormously simplifies managing multiple documents. See “GenDocument,” Chapter 13, for information on the Document Control objects.

These objects are very simple to use. In particular, the GenPrimary is very easy to declare and use. Although all applications use the GenPrimary, many will not need the GenDisplay. However, because **GenPrimaryClass** is a subclass of **GenDisplayClass**, this chapter begins with an overview of **GenDisplayClass**, then continues with a full discussion of **GenPrimaryClass**, and concludes with advanced information about **GenDisplayClass**. Before reading this chapter, you should be familiar with the use and creation of generic user interface objects.

4.1 A First Look at GenDisplay

Not all applications will need to use GenDisplay objects. However, almost all applications will have a GenPrimary object. Since **GenPrimaryClass** is a subclass of **GenDisplayClass**, programmers should be acquainted with **GenDisplayClass**.

This section describes the data fields of **GenDisplayClass** and certain useful messages. It does not have all the information you will need to create these objects. If you will be using GenDisplay objects in your application, you will have to read section 4.3 on page 344.

4.1.1 GenDisplay Object Structure

4.1

The GenDisplay object is a subclass of **GenClass** and therefore inherits all the data fields and attributes of that class. It has few data fields that are set by the application; these fields are listed in Code Display 4-1.

Code Display 4-1 Instance Data of GenDisplayClass

```
/* There are only two instance fields specifically defined for GenDisplayClass.
 * Also, an instance field for GenClass, GI_attrs, has different defaults in
 * GenDisplayClass. */

/* GDI_attributes is a one-byte field for attributes flags. There is only one flag
 * defined for this field, namely GDA_USER_DISMISSABLE, which is on by default. */
    @instance GenDisplayAttrs    GDI_attributes = GDA_USER_DISMISSABLE;

/* The GenDisplay object has a datum for a pointer to a document object. If a
 * Document Control is used to create display objects, it will associate each
 * display with a document object; each will have an optr to the other. */
    @instance optr               GDI_document;

/* The default setting of GI_attrs is different in GenDisplayClass than it is in
 * GenClass: */
    @default GI_attrs = (@default
                       | GA_TARGETABLE
                       | GA_KBD_SEARCH_PATH);

/* The following hints specify whether the display should be minimized or
 * maximized when it is built, and its appearance when minimized. */
    @vardata void HINT_DISPLAY_MINIMIZED_ON_STARTUP;
    @vardata void HINT_DISPLAY_NOT_MINIMIZED_ON_STARTUP;
    @vardata void HINT_DISPLAY_MAXIMIZED_ON_STARTUP;
    @vardata void HINT_DISPLAY_NOT_MAXIMIZED_ON_STARTUP;
    @vardata void HINT_DISPLAY_USE_APPLICATION_MONIKER_WHEN_MINIMIZED;
```

```
/* The following hints and attributes indicate whether the user should be able to
 * minimize, maximize, or resize the window. */
    @vardata void ATTR_GEN_DISPLAY_NOT_MINIMIZABLE;
    @vardata void ATTR_GEN_DISPLAY_NOT_MAXIMIZABLE;
    @vardata void HINT_DISPLAY_NOT_RESIZABLE;

/* ATTR_GEN_DISPLAY_NOT_RESTORABLE indicates that the user should not be able to
 * de-maximize the display once it is maximized. */
    @vardata void ATTR_GEN_DISPLAY_NOT_RESTORABLE;

/* ATTR_GEN_DISPLAY_TRAVELING_OBJECTS is the ChunkHandle of a list of “traveling
 * objects;” these objects are made the children of whichever GenDisplay is on top
 * in a given display region (see section 4.3.3.4 on page 355).*/
    @vardata ChunkHandle ATTR_GEN_DISPLAY_TRAVELING_OBJECTS;

/* The following hints and attributes specify whether the display’s menu bar
 * appears and whether it appears as a “popped out” floating menu. */
    @vardata void HINT_DISPLAY_MENU_BAR_HIDDEN_ON_STARTUP;
    @vardata void TEMP_GEN_DISPLAY_MENU_BAR_HIDDEN;
    @vardata void ATTR_GEN_DISPLAY_MENU_BAR_POPPED_OUT;
    @vardata void HINT_DISPLAY_USE_APPLICATION_MONIKER_WHEN_MENU_BAR_POPPED_OUT;

/* The GenDisplay uses the following vardata fields to store its
 * minimized/maximized state across a shutdown. You should not access these
 * fields. If you want to find out if a GenDisplay is minimized or maximized, send
 * it MSG_GEN_DISPLAY_GET_MINIMIZED or MSG_GEN_DISPLAY_GET_MAXIMIZED. */
    @vardata    void ATTR_GEN_DISPLAY_MINIMIZED_STATE;
    @vardata    void ATTR_GEN_DISPLAY_MAXIMIZED_STATE;

/* HINT_DISPLAY_DEFAULT_ACTION_IS_NAVIGATE_TO_NEXT_FIELD specifies the default
 * action for the GenDisplay. */
    @vardata void HINT_DISPLAY_DEFAULT_ACTION_IS_NAVIGATE_TO_NEXT_FIELD;
```

4.1.1.1 The GDI_attributes Field

MSG_GEN_DISPLAY_GET_ATTRS, MSG_GEN_DISPLAY_SET_ATTRS

The GenDisplay object has a one-byte record called *GDI_attributes* to store attribute flags. There is only one attribute flag, namely GDA_USER_DISMISSABLE. If this attribute is set, the user can dismiss a display through the UI (without choosing a command in the application). Details of this depend on the specific UI; for example, in OSF/Motif, the user could dismiss a display by double-clicking the “Control button.”

■ MSG_GEN_DISPLAY_GET_ATTRS

```
GenDisplayAttrs MSG_GEN_DISPLAY_GET_ATTRS();
```

This message retrieves the *GDI_attributes* field from the destination object.

Source: Unrestricted.

Destination: Any GenDisplay or GenPrimary object.

Return: A **GenDisplayAttrs** record. The only flag defined is GDA_USER_DISMISSABLE.

Interception: This message is not normally intercepted.

■ MSG_GEN_DISPLAY_SET_ATTRS

```
void        MSG_GEN_DISPLAY_SET_ATTRS(
            GenDisplayAttrs     attrs);
```

This message changes the **GenDisplayAttrs** field of the destination object.

Source: Unrestricted.

Destination: Any GenDisplay or GenPrimary object**.**

Parameters: *attrs* Field of **GenDisplayAttrs** flags. There is only one flag defined, namely GDA_USER_DISMISSABLE.

Interception: This message is not normally intercepted.

4.1.1.2 The GDI_document Field

Applications often use the Document Control objects to manage files. With this mechanism, every file is associated with a document object. Often, each file will have its own GenDisplay object as well. In this case, *GDI_document* will contain an optr to the GenDocument object associated with this GenDisplay. For more information on this, see section 4.3 on page 344. The Document Control objects create and destroy the GenDisplays automatically, and set this field accordingly. The GenDisplay object uses this field only when the display is closed; see section 4.3.3.1 on page 353. To retrieve the value of this field, send MSG_GEN_DISPLAY_GET_DOCUMENT to the display (see section 4.3.3.3 on page 354).

4.1.2 Minimizing and Maximizing

```
MSG_GEN_DISPLAY_SET_MINIMIZED,
MSG_GEN_DISPLAY_SET_NOT_MINIMIZED,
MSG_GEN_DISPLAY_GET_MINIMIZED,
MSG_GEN_DISPLAY_SET_MAXIMIZED,
MSG_GEN_DISPLAY_SET_NOT_MAXIMIZED,
MSG_GEN_DISPLAY_GET_MAXIMIZED,
HINT_DISPLAY_MINIMIZED_ON_STARTUP,
HINT_DISPLAY_NOT_MINIMIZED_ON_STARTUP,
HINT_DISPLAY_MAXIMIZED_ON_STARTUP,
HINT_DISPLAY_NOT_MAXIMIZED_ON_STARTUP,
ATTR_GEN_DISPLAY_NOT_MINIMIZABLE,
HINT_DISPLAY_NOT_MAXIMIZABLE,
ATTR_GEN_DISPLAY_NOT_RESTORABLE,
HINT_DISPLAY_DEFAULT_ACTION_IS_NAVIGATE_TO_NEXT_FIELD
```

Windows (i.e. displays and “primary” windows) can be resized by the user. How resizing is done depends on the specific UI; for example, in OSF/Motif, a user resizes a window by dragging its edge. Most specific UIs also allow the user to “minimize” or “maximize” a window. When a window is maximized, it is expanded to fill all available space; that is, a primary window will fill the screen, while a GenDisplay will fill the display area. Windows can also be “minimized.” A window’s behavior when it is minimized depends on the specific UI. For example, in OSF/Motif, a minimized Primary is displayed as an icon at the bottom of the screen; a minimized Display is removed from the display area, but stays in the display control’s display list.

Most of the mechanics of minimizing and maximizing windows is taken care of by the specific UI. For example, OSF/Motif provides minimize and maximize buttons on all Displays and Primaries which do not specifically disable the functionality. However, an application can send messages to Primary and Display objects to change their minimized/maximized state, or to find out what the current state is.

If you do not wish to have a window be minimizable, you can set the vardata flag ATTR_GEN_DISPLAY_NOT_MINIMIZABLE. Similarly, if you do not wish the window to be maximizable, you can set the flag ATTR_GEN_DISPLAY_NOT_MAXIMIZABLE. These instruct the specific UI not to provide the controls for minimizing and maximizing.

You can find out whether a window is currently minimized by sending it MSG_GEN_DISPLAY_GET_MINIMIZED. Similarly, you can find out whether the window is maximized by sending MSG_GEN_DISPLAY_GET_MAXIMIZED.

If a GenDisplay or GenPrimary has the vardata HINT_DISPLAY_MINIMIZED_ON_STARTUP, the object will be created in its minimized state. Similarly, if you set HINT_DISPLAY_NOT_MINIMIZED_ON_STARTUP, the display will be created in its non-minimized form. If you set HINT_DISPLAY_MAXIMIZED_ON_STARTUP, the specific UI will create the object in its maximized state; correspondingly, if you set HINT_DISPLAY_NOT_MAXIMIZED_ON_STARTUP, the specific UI will create the display in a non-maximized state. As with all hints, the specific UI may ignore these directives. If you set conflicting hints (for example, both HINT_DISPLAY_MINIMIZED_ON_STARTUP and HINT_DISPLAY_MAXIMIZED_ON_STARTUP), the results are undefined.

Most displays which can be maximized can also be “restored”; that is, a control is provided which de-maximizes the object, restoring it to the size it was before it was maximized. If the object has the attribute ATTR_GEN_DISPLAY_NOT_RESTORABLE, this control will not be provided; once a display is maximized, the user will not be able to un-maximize it. This hint is generally set only for GenDisplay objects that are maximized on startup.

If you do not want a user to be able to resize a GenDisplay or GenPrimary, set the vardata HINT_DISPLAY_NOT_RESIZABLE. The specific UI will not provide the means for the user to resize the window. This hint will not prevent the user from minimizing or maximizing the display.

If you want the user to be able to navigate from GenDisplays using TAB navigation, add HINT_DISPLAY_DEFAULT_ACTION_IS_NAVIGATE_TO_NEXT_FIELD to the object’s instance data.

■ MSG_GEN_DISPLAY_SET_MINIMIZED

```
void      MSG_GEN_DISPLAY_SET_MINIMIZED();
```

This message instructs a display or primary object to minimize itself. The result depends on the specific UI. Primary windows are usually iconified;

display windows might be iconified or temporarily removed. If the window is already minimized, the message has no effect.

Source: Unrestricted.

Destination: Any GenDisplay or GenPrimary object.

Interception: You should not change the behavior of this message. You may intercept this message to find out when a window is being minimized; however, you should make sure to pass this message on to the superclass.

■ MSG_GEN_DISPLAY_SET_MAXIMIZED

```
void        MSG_GEN_DISPLAY_SET_MAXIMIZED();
```

This message instructs a display or primary object to maximize itself. If the window is already maximized, the message has no effect.

Source: Unrestricted.

Destination: Any GenDisplay or GenPrimary object.

Interception: You should not change the behavior of this message. You may intercept this message to find out when a window is being maximized; however, you should make sure to pass this message on to the superclass.

■ MSG_GEN_DISPLAY_SET_NOT_MINIMIZED

```
void        MSG_GEN_DISPLAY_SET_NOT_MINIMIZED();
```

This message instructs a display or primary object to de-minimize itself. It will generally be restored to its position and configuration as of the time it was minimized. If the window is not minimized, the message has no effect.

Source: Unrestricted.

Destination: Any GenDisplay or GenPrimary object.

Interception: You should not change the behavior of this message. You may intercept this message to find out when a window is being de-minimized; however, you should make sure to pass this message on to the superclass.

■ MSG_GEN_DISPLAY_SET_NOT_MAXIMIZED

```
void        MSG_GEN_DISPLAY_SET_NOT_MAXIMIZED();
```

This message instructs a display or primary object to de-maximize itself. It will generally be restored to its position and configuration as of the time it was maximized. If the window is not maximized, the message has no effect.

Source: Unrestricted.

Destination: Any GenDisplay or GenPrimary object.

Interception: You should not change the behavior of this message. You may intercept this message to find out when a window is being de-maximized; however, you should make sure to pass this message on to the superclass.

■ MSG_GEN_DISPLAY_GET_MINIMIZED

```
Boolean MSG_GEN_DISPLAY_GET_MINIMIZED();
```

This message indicates whether the recipient is minimized.

Source: Unrestricted.

Destination: Any GenDisplay or GenPrimary object.

Return: Returns *true* (i.e. non-zero) if recipient is minimized; otherwise, it returns *false* (i.e. zero).

Interception: You should not intercept this message.

■ MSG_GEN_DISPLAY_GET_MAXIMIZED

```
Boolean MSG_GEN_DISPLAY_GET_MAXIMIZED();
```

This message indicates whether the recipient is maximized.

Source: Unrestricted.

Destination: Any GenDisplay or GenPrimary object.

Return: Returns *true* (i.e. non-zero) if recipient is maximized; otherwise, it returns *false* (i.e. zero).

Interception: You should not intercept this message.

4.2 Using the GenPrimary

MSG_GEN_PRIMARY_GET_LONG_TERM_MONIKER, MSG_GEN_PRIMARY_USE_LONG_TERM_MONIKER, MSG_GEN_PRIMARY_REPLACE_LONG_TERM_MONIKER, HINT_PRIMARY_FULL_SCREEN, HINT_PRIMARY_NO_FILE_MENU,

```
HINT_PRIMARY_NO_EXPRESS_MENU,
HINT_PRIMARY_OPEN_ICON_BOUNDS, HINT_PRIMARY_NO_HELP_BUTTON
```

Almost every application will have a single GenPrimary object. GEOS uses the GenPrimary to create and manage the primary window of an application. A few applications may have several GenPrimary objects; a very few will have no GenPrimary at all. (Applications with no GenPrimary generally do not have any user interface; they often are intended to work with other applications, communicating via streams.)

The structure of the GenPrimary object is almost the same as that of GenDisplay. The instance data definitions for **GenPrimaryClass** are shown in Code Display 4-2 below.

Code Display 4-2 Instance data for GenPrimaryClass

```
/* There is only one instance field specifically defined for GenPrimaryClass. */

/* A GenPrimary object can have a long moniker, which is displayed at the top of
 * the window. The moniker is stored in a chunk in the object block containing the
 * GenPrimary; the attribute GPI_longTermMoniker contains the handle of this
 * chunk. The long-term moniker is described below.*/
    @instance ChunkHandle       GPI_longTermMoniker;

/* GenPrimaryClass also modifies the default GI_attrs settings. */
    @default GI_attrs = @default | GA_TARGETABLE;

/* HINT_PRIMARY_FULL_SCREEN indicates that the primary object should be sized to
 * fill a large portion of the screen. If this hint is not present, the Primary
 * will be just large enough to accommodate its children.*/
    @vardata void       HINT_PRIMARY_FULL_SCREEN;

/* Ordinarily, every Primary window is created with a File menu. You can suppress
 * this by including HINT_PRIMARY_NO_FILE_MENU. */
    @vardata void       HINT_PRIMARY_NO_FILE_MENU;

/* Also by default, any launched Primary window gets added to the Express Menu. If
 * you wish to suppress this behavior, add HINT_PRIMARY_NO_EXPRESS_MENU. */
    @vardata void       HINT_PRIMARY_NO_EXPRESS_MENU;

/* If a primary object is minimizable, the location of the minimized primary is
 * stored in the vardata field HHINT_PRIMARY_OPEN_ICON_BOUNDS.
 */
    @vardata Rectangle  HINT_PRIMARY_OPEN_ICON_BOUNDS
```

```
/* By default, all primary windows have a “help” button; when the user clicks on
 * it, the window’s help text is brought up. If you don’t want the primary to
 * provide help text, you can use the hint HINT_PRIMARY_NO_HELP_BUTTON.
 */
    @vardata void HINT_PRIMARY_NO_HELP_BUTTON;
```

When a Primary window is created, it is usually sized to contain all of its components. However, you can suggest that it be sized to fill almost all the screen by setting the hint HINT_PRIMARY_FULL_SCREEN. This hint says that the Primary should be sized to fill a large portion of the screen, though not all of it. (For example, OSF/Motif sets the Primary to fill the whole screen except for a narrow space for icons at the bottom.) If this hint is not present, the Primary will be just large enough to accommodate its children.

A GenPrimary normally creates a File menu within its menu bar. To suppress creation of this file menu, add HINT_PRIMARY_NO_FILE_MENU. GenPrimarys, by default, are also added to the list of active applications within the system’s express menu. Add HINT_PRIMARY_NO_EXPRESS_MENU if you wish to avoid adding the launched GenPrimary to the express menu.

When a Primary is minimized, it is displayed as an icon with a caption beneath it. The icon and caption will be taken from the Primary’s moniker list (*GI_visMoniker*). If the Primary lacks either a text moniker or a graphic moniker, the missing moniker will be read from the Application object’s *GI_visMoniker* field. Most applications will not set *GI_visMoniker* in the Primary object, since it would usually mean duplicating the monikers already in the Application object. However, some applications will set this (e.g. because they have several Primary objects and want each one to have a different icon when minimized).

When the Primary is expanded from a minimized state, its minimized location is stored in the hint HINT_PRIMARY_OPEN_ICON_BOUNDS. The next time the Primary is minimized, it will be returned to that location.

GPI_longTermMoniker contains a secondary moniker for the Primary object. This moniker is displayed along with the Primary moniker, in a way which depends on the specific UI. (In OSF/Motif, the Primary’s text monikers are shown in its title bar: first the text moniker from *GI_visMoniker*, then a dash, then the moniker from *GPI_longTermMoniker*.) If the GenPrimary has a

GenDisplayGroup as a child, the GenDisplayGroup will automatically set this field to contain the moniker of the top-most GenDisplay. The application can override this by sending the Primary MSG_GEN_PRIMARY_USE_LONG_TERM_MONIKER, described below.

Most Primary objects will have help text; under most specific UIs, the Primary will have a "help" button to bring up this text. If you don't want to provide help text, you should set the hint HINT_PRIMARY_NO_HELP_BUTTON.

■ MSG_GEN_PRIMARY_GET_LONG_TERM_MONIKER

```
ChunkHandle MSG_GEN_PRIMARY_GET_LONG_TERM_MONIKER();
```

Use this message to find out the moniker of a GenPrimary object. The message returns the chunk handle of the moniker; the moniker is in the same block as the GenPrimary object.

Source: Unrestricted.

Destination: GenPrimary.

Return: Returns the chunk handle of the primary's long-term moniker. The chunk is in the same object block as the GenPrimary.

Interception: This message is not ordinarily intercepted.

■ MSG_GEN_PRIMARY_USE_LONG_TERM_MONIKER

```
void        MSG_GEN_PRIMARY_USE_LONG_TERM_MONIKER(
            ChunkHandle         moniker); /* must be in same object block as
                                           * primary */
```

This message instructs a primary window to change its long-term moniker. The new long-term moniker must already be in a chunk in the same object block as the Primary. The chunk containing the obsolete long-term moniker will not be freed; you must do this manually.

Source: Unrestricted.

Destination: GenPrimary.

Parameters: *moniker* ChunkHandle of chunk in same object block as the GenPrimary. The chunk should contain the new long-term moniker.

Interception: This message is not ordinarily intercepted.

■ MSG_GEN_PRIMARY_REPLACE_LONG_TERM_MONIKER

```
void    MSG_GEN_PRIMARY_REPLACE_LONG_TERM_MONIKER(@stack
        VisUpdateMode       updateMode,
        word                height,
        word                width,
        word                length,
        VisMonikerDataType  dataType,
        VisMonikerSourceType sourceType,
        dword               source);
```

This message is used to replace a primary's long term moniker with a new one. This message's arguments are precisely like those to the message MSG_GEN_REPLACE_VIS_MONIKER. Note that a long term moniker is ordinarily a simple text string. For more information, see "Managing Visual Monikers", section 2.4.2 of chapter 2.

Source: Unrestricted.

Destination: GenPrimary.

Parameters: The parameters are the same as those for MSG_GEN_REPLACE_VIS_MONIKER.

Return: Returns the chunk handle of the new long-term moniker. The moniker will be allocated in the Primary's object block.

Interception: This message is not ordinarily intercepted.

4.3 Using Multiple Displays

Many applications will need to have several similar user interface areas. For example, a word processor might have several documents open at once; each of these would need its own text area. GEOS provides a facility for this.

An application can have several GenDisplay objects, each of which must be a child of a GenDisplayGroup object. The user can then switch back and forth between the displays using the GenDisplayControl (which is usually a child of a "Window" menu). The switching is transparent to the application.

If the application uses one display for each document, it should use the Document Control objects to create the displays. The Document Control can automatically duplicate a resource containing a generic object tree headed by a GenDisplay and make that GenDisplay a child of the GenDisplayGroup each time a document is opened or created. For details, see “GenDocument,” Chapter 13.

4.3.1 GenDisplayGroup

If an application uses GenDisplay objects, it must have a GenDisplayGroup object. This object makes sure there is space in the GenPrimary for the displays.

The GenDisplayGroup must be either a child of the GenPrimary (in which case the specific UI will decide where to put the display area) or a child of a **GenInteraction** which is a child of the GenPrimary (if the application wants the display area in a specific part of the GenPrimary). The GenDisplayGroup should be run by the UI thread.

GenDisplayGroupClass has no instance data which may be set or examined by the application. However, **GenDisplayGroupClass** is a subclass of **GenClass**, and inherits all of its instance data. When you declare a GenDisplayGroup, you may specify its **GenClass** instance data normally; you may also include any of the hints described in the following sections.

4.3.1.1 The GenDisplayGroup Instance Data

```
HINT_DISPLAY_GROUP_SEPARATE_MENUS,
HINT_DISPLAY_GROUP_ARRANGE_TILED,
HINT_DISPLAY_GROUP_FULL_SIZED_ON_STARTUP,
HINT_DISPLAY_GROUP_OVERLAPPING_ON_STARTUP,
HINT_DISPLAY_GROUP_FULL_SIZED_IF_TRANSPARENT_DOC_CTRL_MODE
, HINT_DISPLAY_GROUP_TILE_HORIZONTALLY,
HINT_DISPLAY_GROUP_TILE_VERTICALLY,
HINT_DISPLAY_GROUP_SIZE_INDEPENDENTLY_OF_DISPLAYS
```

GenDisplayGroupClass is a subclass of GenClass. Other than vardata, this class adds no other instance data. There are several hints defined for **GenDisplayGroupClass**. Most of these hints specify how displays should be arranged on startup.

Code Display 4-3 GenDisplayGroup Instance Data

```
/* GenDisplayGroupClass adds no instance fields. It does modify the default
 * GI_attrs settings, however. */
    @default GI_attrs = @default | GA_TARGETABLE;

/* This hint allows each GenDisplay to contain its own menu bar. */
    @vardata void       HINT_DISPLAY_GROUP_SEPARATE_MENUS;

/* These hints specify how a GenDisplayGroup will arrange its GenDisplays. */
    @vardata void       HINT_DISPLAY_GROUP_ARRANGE_TILED;
    @vardata void       HINT_DISPLAY_GROUP_FULL_SIZED_ON_STARTUP;
    @vardata void       HINT_DISPLAY_GROUP_OVERLAPPING_ON_STARTUP;
    @vardata void
HINT_DISPLAY_GROUP_FULL_SIZED_IF_TRANSPARENT_DOC_CTRL_MODE;
    @vardata void       HINT_DISPLAY_GROUP_TILE_HORIZONTALLY;
    @vardata void       HINT_DISPLAY_GROUP_TILE_VERTICALLY;
    @vardata void       HINT_DISPLAY_GROUP_SIZE_INDEPENDENTLY_OF_DISPLAYS;

/* These attributes affect the availability of overlapping and
 * full-sized states. */
    @vardata void       ATTR_GEN_DISPLAY_GROUP_NO_FULL_SIZED;
    @vardata void       ATTR_GEN_DISPLAY_GROUP_NO_OVERLAPPING;
    @vardata void       ATTR_GEN_DISPLAY_OVERLAPPING_STATE;
```

4.3

In some specific UIs (such as OSF/Motif), menus which are children of a GenDisplay object may appear in two ways: they may be drawn on the Primary's menu bar (the default in OSF/Motif), or they may appear in a menu bar on the display itself. HINT_DISPLAY_GROUP_SEPARATE_MENUS indicates that each display should be given its own menu bar (if the specific UI permits this).

There are several hints which specify how the displays should be configured when the GenDisplayGroup is built.

HINT_DISPLAY_GROUP_FULL_SIZED_ON_STARTUP indicates that the GenDisplayGroup should be in full-size mode on startup; that is, all of its children should be maximized.

HINT_DISPLAY_GROUP_OVERLAPPING_ON_STARTUP indicates that the GenDisplayGroup should be in overlapping mode on startup; that is, its children should be non-maximized.

HINT_DISPLAY_GROUP_FULL_SIZED_IF_TRANSPARENT_DOC_CTRL_MODE forces a GenDisplayGroup to start full-sized if the application is in "transparent document control" mode, which is set by the user level of the application. This hint overrides HINT_DISPLAY_GROUP_OVERLAPPING_ON_STARTUP, if present.

HINT_DISPLAY_GROUP_ARRANGE_TILED indicates that the GenDisplayGroup should be in overlapping mode on startup, and further that the displays should be tiled; that is, they should be non-maximized and arranged in a non-overlapping way to fill the display area.

You can specify a preference for how the displays should be tiled by setting HINT_DISPLAY_GROUP_TILE_HORIZONTALLY or HINT_DISPLAY_GROUP_TILE_VERTICALLY. HINT_DISPLAY_GROUP_TILE_HORIZONTALLY indicates that you want tiled displays to be arranged horizontally, with each display tall enough to fill the display area. Similarly, HINT_DISPLAY_GROUP_TILE_VERTICALLY indicates that you want tiled displays to be arranged vertically, with each display wide enough to fill the display area. If both hints are set, the result varies depending on the specific UI.

HINT_DISPLAY_GROUP_SIZE_INDEPENDENTLY_OF_DISPLAYS sizes a GenDisplayGroup by what its parent wants rather than what any of its children GenDisplays want. This may improve geometry performance in a complex GenPrimary/GenDisplay combination.

4.3.1.2 Arranging Displays in the Display Group

```
MSG_GEN_DISPLAY_GROUP_SET_OVERLAPPING,
MSG_GEN_DISPLAY_GROUP_SET_FULL_SIZED,
MSG_GEN_DISPLAY_GROUP_GET_FULL_SIZED,
MSG_GEN_DISPLAY_GROUP_TILE_DISPLAYS,
ATTR_GEN_DISPLAY_GROUP_NO_FULL_SIZED,
ATTR_GEN_DISPLAY_GROUP_NO_OVERLAPPING,
ATTR_GEN_DISPLAY_GROUP_OVERLAPPING_STATE
```

The GenDisplayGroup can be in "full-sized" or "overlapping" mode. If the GenDisplayGroup is in "full-size" mode, all of its children are maximized (except any displays which are set "non-maximizable"). If it is not in full-sized mode, it is said to be in "overlapping" mode; that is, none of its children are maximized. When a user maximizes any display which belongs to a GenDisplayGroup, the GenDisplayGroup automatically goes into “full-size” mode and maximizes all of its children.

If you include ATTR_GEN_DISPLAY_GROUP_NO_FULL_SIZED, the GenDisplayControl will not be able to go into full-size mode; it will always be in overlapping mode. Similarly, if set ATTR_GEN_DISPLAY_GROUP_NO_OVERLAPPING, the GenDisplayControl will not be able to go into overlapping mode; it will always be in full-sized mode, and all displays will always be maximized. Naturally, you may not include both of these attributes at once; if you do, results are undefined.

Messages are provided which switch the GenDisplayGroup into one or another of these modes. You might not need to use any of these messages. If you use a GenDisplayControl object, the user will be able to switch from overlapping to full-size and also to tile the displays by using that object. However, you can also send the following messages directly.

You can set a GenDisplayGroup to full-sized mode by sending it the message MSG_GEN_DISPLAY_GROUP_SET_FULL_SIZED. This message causes the GenDisplayGroup to maximize every one of its children. Children which cannot be maximized will be unaffected. The window layering and focus/target settings are not changed.

You can set a GenDisplayGroup to overlapping mode by sending it the message MSG_GEN_DISPLAY_GROUP_SET_OVERLAPPING. This message causes a GenDisplayGroup object to de-maximize all of its children. Children which are not restorable will be unaffected. The window layering and focus/target settings are not changed.

You can find out whether a GenDisplayGroup object is in full-sized mode by sending it MSG_GEN_DISPLAY_GROUP_GET_FULL_SIZED. If the GenDisplayGroup is in full-sized mode, this message will return *true*.

You can also put a GenDisplayGroup into “tiled” mode. This is a special case of overlapping mode. When a GenDisplayGroup is put in tiled mode, it first puts itself in overlapping mode. It then attempts to arrange and resize its display children so they fill the display area without overlapping. To put a GenDisplayGroup into tiled mode, send it the message MSG_GEN_DISPLAY_GROUP_TILE_DISPLAYS.

The GenDisplayGroup keeps track of its overlapping state across shutdowns. It does this by setting ATTR_GEN_DISPLAY_GROUP_OVERLAPPING_STATE. Applications may not set or change this attribute directly.

■ MSG_GEN_DISPLAY_GROUP_SET_FULL_SIZED

```
void        MSG_GEN_DISPLAY_GROUP_SET_FULL_SIZED();
```

This message instructs a GenDisplayGroup to put itself in “full-sized” mode; that is, all of its maximizable children will be maximized. This message is ignored if the GenDisplayGroup has the vardata attribute ATTR_GEN_DISPLAY_GROUP_NO_FULL_SIZED.

Source: Unrestricted.

Destination: GenDisplayGroup.

Interception: Not generally intercepted.

■ MSG_GEN_DISPLAY_GROUP_SET_OVERLAPPING

```
void        MSG_GEN_DISPLAY_GROUP_SET_OVERLAPPING();
```

This message instructs a GenDisplayGroup to put itself in “overlapping” mode. This message is ignored if the GenDisplayGroup has the vardata attribute ATTR_GEN_DISPLAY_GROUP_NO_OVERLAPPING.

Source: Unrestricted.

Destination: GenDisplayGroup.

Interception: Not generally intercepted.

■ MSG_GEN_DISPLAY_GROUP_GET_FULL_SIZED

```
Boolean     MSG_GEN_DISPLAY_GROUP_GET_FULL_SIZED();
```

This message finds out whether a GenDisplayGroup is in “full-sized” mode.

Source: Unrestricted.

Destination: GenDisplayGroup.

Return: Returns *true* (i.e. non-zero) if the GenDisplayGroup is in full-sized mode; otherwise, it returns *false* (i.e. zero).

Interception: Not generally intercepted.

■ MSG_GEN_DISPLAY_GROUP_TILE_DISPLAYS

```
void      MSG_GEN_DISPLAY_GROUP_TILE_DISPLAYS;
```

This message instructs a GenDisplayGroup to put itself in “tiled” mode. That is, it should first put itself in “overlapping” mode; it should then arrange and resize the displays so they fill the display area without overlapping. The message is ignored if the GenDisplayGroup has the vardata attribute ATTR_GEN_DISPLAY_GROUP_NO_OVERLAPPING.

Source: Unrestricted.

Destination: GenDisplayGroup.

Interception: Not generally intercepted.

4.3.1.3 Selecting a Display

MSG_GEN_DISPLAY_GROUP_SELECT_DISPLAY

Ordinarily, the user switches from one display to another in one of two ways. The user may use the specific UI’s way of switching displays (e.g. clicking on the display); or he may use the GenDisplayControl (described below) to switch displays. The application can also force the Display Group to bring a certain display to the top by sending it MSG_GEN_DISPLAY_GROUP_SELECT_DISPLAY. However, this is not usually done; applications should generally let the user switch displays with the GenDisplayControl.

■ MSG_GEN_DISPLAY_GROUP_SELECT_DISPLAY

```
void      MSG_GEN_DISPLAY_GROUP_SELECT_DISPLAY(
          word   displayNum);
```

This message instructs a display group to select a certain display, bringing it to the top and making it the focus. Applications should not ordinarily need to send this.

Source: Usually GenDisplayControl or its associated objects; however, any object can send this.

Destination: GenDisplayGroup.

Parameters: *displayNum* The display to select. This is an integer specifying the position of the desired display among the GenDisplayGroup’s children; that is, its first child is number zero, its next child is number one, and so on.

Interception: This message is not ordinarily intercepted.

Tips: You can find a display's position number by sending MSG_GEN_FIND_CHILD to the GenDisplayGroup, passing the optr of the desired display. See section 2.5.1 of chapter 2.

4.3.2 GenDisplayControl

The GenDisplayGroup object does half the job of managing display objects: it creates a space for the displays and manages them as its children. However, the display group does very little interaction with the user. Instead, the user works mainly with the GenDisplayControl object.

The GenDisplayControl is usually a child of a “Window” GenInteraction, which is itself a child of the primary. **GenDisplayControlClass** is a subclass of **GenControlClass**, and it has all the functionality of that class. For more information, see “Generic UI Controllers,” Chapter 12.

GenDisplayControlClass is based very closely on **GenControlClass**. The differences between it and **GenControlClass** are shown below in Code Display 4-4.

Code Display 4-4 Instance Data of GenDisplayControlClass

```
/* GDCII_attrs is a byte-length flag field. There is currently only one flag,
 * namely GDCA_MAXIMIZED_NAME_ON_PRIMARY; this specifies that if the active
 * display is maximized, its name should be shown as the primary’s long-term
 * moniker. */
    @instance   GenDisplayControlAttributes     GDCII_attrs =
                                                (GDCA_MAXIMIZED_NAME_ON_PRIMARY);

@default        GI_states = @default | GS_ENABLED;
@default        GCI_output = {TO_APP_TARGET};

typedef WordFlags GDCFeatures;
#define GDCF_OVERLAPPING_MAXIMIZED       0x0004
#define GDCF_TILE                        0x0002
#define GDCF_DISPLAY_LIST                0x0001
```

```
typedef WordFlags GDCToolboxFeatures;
#define GDCTF_OVERLAPPING_MAXIMIZED      0x0004
#define GDCTF_TILE                       0x0002
#define GDCTF_DISPLAY_LIST               0x0001

#define GDC_DEFAULT_FEATURES     (GDCF_OVERLAPPING_MAXIMIZED | GDCF_TILE
                                 | GDCF_DISPLAY_LIST)

#define GDC_DEFAULT_TOOLBOX_FEATURES (GDCF_DISPLAY_LIST)

/* A GenDisplayControl also features a single hint which affects the display of
 * the features list. */
    @vardata void HINT_DISPLAY_CONTROL_NO_FEATURES_IF_TRANSPARENT_DOC_CTRL_MODE;
```

The *GDCII_attrs* field contains a set of **GenDisplayControlAttributes**. There is currently only one flag defined among these attributes:

GDCA_MAXIMIZED_NAME_ON_PRIMARY
If this attribute is set and the active display is maximized, the name of the selected display will be shown in the long term moniker of the GenPrimary.

HINT_DISPLAY_CONTROL_NO_FEATURES_IF_TRANSPARENT_DOC_CTRL_MODE suppresses display of the features list if the application is running in “transparent document control” mode, as selected by the user level of the application.

4.3.3 Using GenDisplayClass Objects

All GenDisplay objects must be children of a GenDisplayGroup object. GenDisplay objects can be created in several ways: an application can declare them in its code; it can instantiate them at run-time and make them children of the GenDisplayGroup; or, if the application uses the Document Control objects, it can have the Document Control create a new display automatically whenever a document is opened. (For details about using a Document Control to create GenDisplay objects, see “GenDocument,” Chapter 13.)

4.3.3.1 Closing GenDisplays

MSG_GEN_DISPLAY_CLOSE

Most specific UIs provide a way for the user to close windows. For example, OSF/Motif lets a user close a window by double-clicking the control button. When the user uses the specific UI's close mechanism, the Display or Primary is sent MSG_GEN_DISPLAY_CLOSE.

GenDisplayClass does only one thing when it receives MSG_GEN_DISPLAY_CLOSE: it sends MSG_GEN_DOCUMENT_CLOSE to the document specified by *GDI_document*. The **GenDisplayClass** handler for MSG_GEN_DISPLAY_CLOSE does not, in fact, destroy the display. If the display is linked to a GenDocument, the GenDocument will respond to MSG_GEN_DOCUMENT_CLOSE by closing the application and removing the GenDisplay. Otherwise, you will have to remove the GenDisplay yourself by writing a handler for MSG_GEN_DISPLAY_CLOSE.

The **GenPrimaryClass** handler for MSG_GEN_DISPLAY_CLOSE closes the application. If you want to add to or replace this behavior, you may have your Primary subclass this message.

■ MSG_GEN_DISPLAY_CLOSE

```
void        MSG_GEN_DISPLAY_CLOSE();
```

This message is sent to a Display to close it. The system sends it when the user uses the specific UI's way of closing a window. The **GenDisplayClass** handler does nothing but send a MSG_GEN_DOCUMENT_CLOSE to the Document object specified in *GDI_document*. The **GenPrimaryClass** handler closes the application.

Source: Unrestricted.

Destination: GenDisplay.

Interception: If the Display is not associated with a GenDocument, you will need to subclass this message for it to have any effect at all. If the display is associated with a GenDocument object, you should probably subclass your Document object's MSG_GEN_DOCUMENT_CLOSE instead. Primary objects may subclass this message if they want to alter or replace the default behavior (of closing the application).

4.3.3.2 Menu Bar PopOuts

```
ATTR_GEN_DISPLAY_MENU_BAR_POPPED_OUT
```

Some objects contain the ability to "pop out" of their sub-group locations and become floating dialog boxes.The menu bar of a GenDisplay is one such GenInteraction. If the menu bar of a GenDisplay is currently in the "popped-out" state, it will contain the vardata entry ATTR_GEN_DISPLAY_MENU_BAR_POPPED_OUT.

4.3.3.3 Messages sent to GenDisplays

```
MSG_GEN_DISPLAY_UPDATE_FROM_DOCUMENT,
MSG_GEN_DISPLAY_GET_DOCUMENT
```

Many of the messages which can be sent to GenDisplay objects have already been discussed above in section 4.1. However, there are a few messages which are ordinarily sent to Displays but not to Primaries. These messages are discussed here.

GenDisplay objects often work in close conjunction with the Document Control objects. It is common for every open document to have its own GenDisplay object as well as its own GenDocument. The two objects work in conjunction, sending messages back and forth to communicate. You can send or intercept these messages yourself to add functionality.

When the document object changes in certain significant ways, the Display has to be brought into accord with it. For example, if the name of the file changes, the GenDisplay's moniker will have to be changed to reflect this. Whenever a significant change takes place, the Document Control objects send a MSG_GEN_DISPLAY_UPDATE_FROM_DOCUMENT to the appropriate Display. The Display then requests all necessary information from the GenDocument and makes any necessary changes to its own instance data. You can force this updating by sending the message directly to the Display. You can also subclass this message if you want to add special updating behavior; however, you should be sure to pass this message to the superclass' handler.

You can find out which Document object is associated with a given Display by sending MSG_GEN_DISPLAY_GET_DOCUMENT to the Display. The message will return an optr to the corresponding Document object.

■ MSG_GEN_DISPLAY_UPDATE_FROM_DOCUMENT

```
void        MSG_GEN_DISPLAY_UPDATE_FROM_DOCUMENT();
```

This message instructs a GenDisplay to update its instance data from its associated GenDocument object (if any). This message is ordinarily sent by the Document Control objects.

Source: Unrestricted—ordinarily sent only by Document Control objects.

Destination: GenDisplay.

Interception: Normally not intercepted. If you subclass this message to add special updating behavior, be sure to end with an **@callsuper**.

■ MSG_GEN_DISPLAY_GET_DOCUMENT

```
optr        MSG_GEN_DISPLAY_GET_DOCUMENT();
```

This message returns the optr of the GenDocument associated with a given GenDisplay. This is equal to the value of the GenDisplay's *GDI_document* field.

Source: Unrestricted—ordinarily sent only by Document Control objects.

Destination: GenDisplay.

Interception: Normally not intercepted.

4.3.3.4 Traveling Objects

```
ATTR_GEN_DISPLAY_TRAVELLING_OBJECTS
```

If you use multiple GenDisplay objects, it is sometimes useful to set up a group of "traveling objects." Traveling objects are children of whichever display is active. When a different GenDisplay is brought to the top, all traveling objects will be set "not usable" and removed from the Generic tree. They will then be added as children of the new top display and set "usable." (Any children of the traveling objects will naturally move with them.) Traveling objects are most commonly Toolbox Interactions, but they can be any kind of generic object.

Traveling objects can only be used under certain circumstances. Every display must belong to its own object block, and all of these object blocks must be copies of the same original. This is because the traveling objects are added as children to a specified chunk in whichever object block contains the new top display. If you want to use traveling objects, you should declare a special "template" object block which contains a GenDisplay and its children. Whenever you need to create a GenDisplay, you should duplicate this object block. The traveling objects should be in another resource altogether. If you use the Document Control objects to create displays, the objects will use this technique, duplicating an object block for each new display; this will let you use traveling objects.

4.3 Every traveling object is indicated by a **TravelingObjectReference** structure (see Code Display 4-5). To attach traveling objects to the active display, create a chunk which contains a list of **TravelingObjectReference** structures; this chunk must be in the same object block as the active display. Then set the Display's ATTR_GEN_DISPLAY_TRAVELING_OBJECTS to the ChunkHandle of the list. The list will automatically be moved to the block of the active display whenever the traveling objects are moved.

Code Display 4-5 TravelingObjectReference

```
typedef struct {
    optr        TIR_travelingObject;    /* optr to traveling object whose
                                         * reference this is */

    ChunkHandle TIR_parent;             /* Chunk Handle of object in Display's
                                         * block that will be the parent of this
                                         * object */

    word        TIR_compChildFlags;     /* CompChildFlags to use when
                                         * adding the traveling object */
} TravelingObjectReference;
```

The **TravelingObjectReference** structure has the following three fields:

TIR_travelingObject
This field is an optr to the traveling object whose reference this is.

TIR_parent This field holds the chunk handle of an object in the display block. When the traveling object is added to a display block, it will be made a child of the object whose chunk handle this is.

Objects

TIR_compChildFlags

This is the set of **CompChildFlags** which will be used when attaching this object to its new parent.

4.3

◆Objects

GenTrigger

5

The GenTrigger object allows a user to initiate an action. In most graphical user interfaces, a GenTrigger is represented by a button; clicking on the button initiates the action. This action sends a pre-specified message to a pre-specified object or process in the trigger's instance data. You may also customize a GenTrigger to pass data along with the message.

5.1

You should be familiar with the function of generic objects in general before you begin reading this chapter. This can be accomplished by a cursory review of "GenClass," Chapter 2. You should at least read "First Steps: Hello World," Chapter 4 of the Concepts Book.

5.1 GenTrigger Overview

Whenever you need to allow the user to initiate a single action, you should probably use a GenTrigger. When activated, a GenTrigger simply sends a message to another object or process. Both the message and its destination can be set up in the GenTrigger's instance data. The trigger can also send data with the message. This is useful if several triggers send the same message (for example MSG_CHOOSE_COLOR) but send different data (for example "Blue," "Red," or "Green").

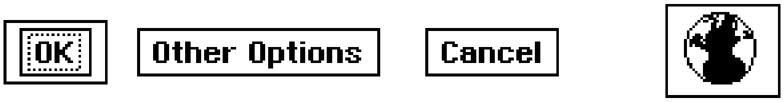


Figure 5-1 *Example GenTriggers*
A trigger's visual moniker can be either text or a graphics string.

In some cases, the use of triggers is readily apparent (the "OK" button in a dialog box, for example, which will dismiss the dialog box) while at other times the use of triggers is not as obvious (as menu items, for example). The GenTrigger is a simple and widely used object, appearing in menus, dialog boxes, and primary windows—wherever you need a UI object whose sole action is to send out a message, a GenTrigger will probably suffice.

Several GenTriggers are shown in Figure 5-1. Because of the trigger's basic role, they cannot have children. Note that this does not mean that activating a trigger may not necessarily bring up an object indirectly (such as a dialog box), but only that an object may not be directly attached below a GenTrigger in a generic object tree.

Your application will choose the function of the trigger but it is left to the Specific UI to choose in what manner a trigger will be activated. (For example, under OSF/Motif, a GenTrigger may be built as a button and will be activated by clicking on the button.)

GenTriggerClass provides two instance fields. These instance fields are listed in Code Display 5-1.

Code Display 5-1 GenTrigger Instance Fields

```
/* GenTrigger inherits all the instance fields of GenClass. */

    @instance optr GTI_destination = 0;
    @instance Message GTI_actionMsg = 0;

@vardata void HINT_TRIGGER_BRINGS_UP_WINDOW;
@vardata void ATTR_GEN_TRIGGER_IMMEDIATE_ACTION;
@vardata void HINT_TRIGGER_DESTRUCTIVE_ACTION;
@vardata word ATTR_GEN_TRIGGER_ACTION_DATA;
@vardata word ATTR_GEN_TRIGGER_INTERACTION_COMMAND;
@vardata Message ATTR_GEN_TRIGGER_CUSTOM_DOUBLE_PRESS;

    /* Alias for messages with “void (word a, word b)” */
typedef struct { word a, b; } TwoWordArgs;
@vardataAlias (ATTR_GEN_TRIGGER_ACTION_DATA)
                TwoWordArgs ATTR_GEN_TRIGGER_ACTION_TWO_WORDS;

    /* Alias for messages with “void (word a, word b, word c)” */
typedef struct { word a, b, c; } ThreeWordArgs;
@vardataAlias (ATTR_GEN_TRIGGER_ACTION_DATA)
                ThreeWordArgs ATTR_GEN_TRIGGER_ACTION_THREE_WORDS;
    /* Alias for messages with “void (optr output, word a)” */
typedef struct { optr output; word a; } OptrWordArgs;
@vardataAlias (ATTR_GEN_TRIGGER_ACTION_DATA)
                        OptrWordArgs ATTR_GEN_TRIGGER_ACTION_OPTR_AND_WORD;
```

The *GTI_actionMsg* instance field sets the message for the trigger to send out upon activation. The activation itself is determined by the specific UI. The trigger will send this message to the output specified in the *GTI_destination* instance field.

5.2

The *GTI_destination* instance field sets the object or process for the trigger to send its message to upon activation. The trigger will send the message in the *GTI_actionMsg* field to this optr. Often this will be a **TravelOption** or a **GenTravelOption**.

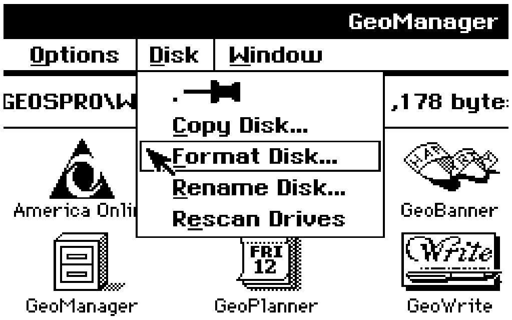


Figure 5-2 *GenTriggers as Menu Items.*
GenTriggers may become menu items within a GIV_POPUP GenInteraction.

5.2 GenTrigger Usage

A simple GenTrigger should contain a message to send out upon activation, a destination to send the message to, and a visual moniker to distinguish the trigger from other objects (and describe its function to the user).

You may use the prototype GEN_TRIGGER_ACTION to define the action message. This prototype passes the optr of the sending trigger in the variable name 'trigger' and can be accessed within a message handler.

GenTriggers may appear within GenPrimarys, GenViews, or, most often, within a GenInteraction. If a GenTrigger is placed as a child of a GIV_POPUP GenInteraction, the GenTriggers may be represented as menu items. (See Figure 5-2).

Code Display 5-2 Code for Basic GenTriggers

```
/* Use the prototype GEN_TRIGGER_ACTION to define a GenTrigger's action message.
 * Within a method, you can retrieve this optr with the variable name 'trigger.' */

@message (GEN_TRIGGER_ACTION) MSG_HELLO_CHANGE_TO_BLUE();
@message (GEN_TRIGGER_ACTION) MSG_HELLO_CHANGE_TO_GOLD();

/* The GenInteraction object will contain the child GenTriggers. */

@object GenInteractionClass HelloColorBox = {
    GI_comp = @HelloBlueTrigger, @HelloGoldTrigger;
    GI_visMoniker = 'C', "Color";
    GII_visibility = GIV_DIALOG;
}

/* GenTriggers
 * Buttons are implemented by GenTriggerClass. When the trigger is pushed by
 * the user (clicked on with the mouse), it will send the specified message
 * to the specified destination object. In both cases below, the trigger will
 * send an application-defined message to the application's Process
 * object. A trigger's moniker is displayed within the trigger. In this
 * case both are text, but any graphics string could also be used to create
 * graphical triggers. */

@object GenTriggerClass HelloBlueTrigger = {
    GI_visMoniker = 'B', "Blue";
        /* Send MSG_HELLO_CHANGE_TO_BLUE to the Process object. */
    GTI_destination = process;
    GTI_actionMsg = MSG_HELLO_CHANGE_TO_BLUE;
}
```

```
@object GenTriggerClass HelloGoldTrigger = {
    GI_visMoniker = ’G’, “Gold”;
        /* Send MSG_HELLO_CHANGE_TO_GOLD to the Process object. */
    GTI_destination = process;
    GTI_actionMsg = MSG_HELLO_CHANGE_TO_GOLD;
}
```

5.3 Supplemental GenTrigger Usage

GenTriggers may also perform other actions besides the simple sending of a message to an object.

For example, your trigger may pass data along with its message; you may then use this data within a method to perform some additional work. In some cases, this data may be interpreted by a GenInteraction object to provide additional functionality to dialog boxes.

You may also alter the manner in which a trigger is initiated through the implementation of hints and several messages. You can also modify the trigger’s instance data dynamically, reassigning its action message and destination.

5.3.1 Passing Data with a GenTrigger

ATTR_GEN_TRIGGER_ACTION_DATA,
ATTR_GEN_TRIGGER_ACTION_TWO_WORDS,
ATTR_GEN_TRIGGER_ACTION_THREE_WORDS,
ATTR_GEN_TRIGGER_ACTION_OPTR_AND_WORD

A GenTrigger may pass data along with its message. Use ATTR_GEN_TRIGGER_ACTION_DATA to assign a word of data that you wish to pass with the particular message. If you need to pass more than a single word of data, you will have to use **@vardataAlias**; this Goc keyword allows

ATTR_GEN_TRIGGER_ACTION_DATA to pass a structure instead of a single word.

Several ATTR_GEN_TRIGGER_ACTION_DATA aliases already exist for your use. ATTR_GEN_TRIGGER_ACTION_TWO_WORDS and ATTR_GEN_TRIGGER_ACTION_THREE_WORDS allow you to pass two or three words of data along with the trigger's message. ATTR_GEN_TRIGGER_ACTION_OPTR_AND_WORD allows you to pass an optr and a word of data along with the trigger's message. 5.3

If you need to pass more than three words of data with a message, you will need to use the stack to pass arguments. To do so, use the **@stack** parameter within your message definition; because of the implementation of the stack, make sure to set up your structure to pass its arguments *in reverse order*.

Code Display 5-3 Passing Data from a GenTrigger

```
/* In a class declaration, you should define a message that passes data. */

@class MyProcessClass, GenProcessClass;
@message void MSG_SET_MY_DATA(int myData);
@endc;

@object GenTriggerClass MyHundredTrigger = {
        /* This trigger will pass 100 to the method for MSG_SET_MY_DATA. */
    GI_visMoniker = “Set My Data to 100”;
    GTI_actionMsg = MSG_SET_MY_DATA;
    GTI_destination = process;
    ATTR_GEN_TRIGGER_ACTION_DATA = 100;  /* This object’s data is 100. */
}

@object GenTriggerClass MyTenTrigger = {
        /* This trigger will pass ‘10’ to the method for MSG_SET_MY_DATA.*/
    GI_visMoniker = “Set My Data to 10”;
    GTI_actionMsg = MSG_SET_MY_DATA;
    GTI_destination = process;
    ATTR_GEN_TRIGGER_ACTION_DATA = 10;   /* This object’s data is 10. */
}

@method MyProcessClass, MSG_SET_MY_DATA {
        /* This message passes one parameter (the integer myData). Goc knows
         * that the data is located within the ATTR_GEN_TRIGGER_ACTION_DATA
         * field. */
    UpdateDisplay(myData);      /* Use that data for your own purposes. */
}
```

```
/* The following examples show how to pass longer structures than the above. */

        /* Define the message. */
@message void MSG_CUSTOM_MESSAGE(@stack char name[10], optr arg1, int arg2);
        /* Define the structure you wish to pass with the message. If the
         * message will pass parameters on the stack (as in this case), you
         * must define your structure to pass its elements in reverse order. */
typedef struct {
    int         MS_arg2;
    optr        MS_arg1;
    char        MS_name[10]
} MyStruct;

        /* Use @vardataAlias to define your own attribute (ATTR_MY_STRUCT_TO_PASS)
         * to store the custom structure. */
@vardataAlias (ATTR_GEN_TRIGGER_ACTION_DATA) MyStruct ATTR_MY_STRUCT_TO_PASS;

        /* Declare your object and set the arguments to pass. */
@object GenTriggerClass MyTrigger = {
    GI_visMoniker = “Custom Trigger”;
    GTI_actionMsg = MSG_CUSTOM_MESSAGE;
    GTI_destination = process;
    ATTR_MY_STRUCT_TO_PASS = { 100, @ListObject, “Zow!” }
}
```

5.3.2 Interaction Commands

ATTR_GEN_TRIGGER_INTERACTION_COMMAND

A GenTrigger may perform special actions within a GenInteraction object. The ATTR_GEN_TRIGGER_INTERACTION_COMMAND attribute sets the special activity for these types of triggers. This attribute indicates that the GenTrigger is an Interaction command trigger for the parent GenInteraction object. ATTR_GEN_TRIGGER_INTERACTION_COMMAND takes a word of data (of type **InteractionCommand**) specifying the command type of the trigger. A trigger with an **InteractionCommand** can perform one of several standard actions. Most of these actions are specific to dialog boxes.

The **InteractionCommand** types are

IC_NULL This command is special, alerting an application that the interaction containing the trigger has been closed because the system is about to shut down. You should never set any triggers to this **InteractionCommand**.

IC_DISMISS This command instructs the UI to dismiss the parent GenInteraction. Set this if you want the trigger to dismiss an Interaction or provide a custom “Close” trigger.

IC_INTERACTION_COMPLETE
This command is special, indicating to the UI that a single interaction has been completed (the user has clicked on an **InteractionCommand** trigger marked with GA_SIGNAL_INTERACTION_COMPLETE). You should never set any triggers to this **InteractionCommand**. If you wish a trigger to exhibit this behavior, set its *GI_attrs* field to include GA_SIGNAL_INTERACTION_COMPLETE. (See section 2.2.1 of chapter 2.)

IC_APPLY This command instructs the UI to apply any properties within a GIT_PROPERTIES Interaction. By default, this trigger will send out a MSG_GEN_APPLY.

IC_RESET This command instructs the UI that this is a “reset” trigger for a properties Interaction; the application is responsible for actual reset behavior. You should provide a method to handle the message sent out by this trigger, set up the state of items to reset, and call MSG_GEN_RESET within that handler.

IC_OK This command instructs the UI that the user has acknowledged a GIT_NOTIFICATION and should take any appropriate measures.

IC_YES This command instructs the UI that the user has signalled a positive response from a GIT_AFFIRMATION Interaction.

IC_NO This command instructs the UI that the user has signalled a negative response from a GIT_AFFIRMATION Interaction.

IC_STOP This command instructs the UI to halt a GIT_PROGRESS Interaction; (the application is responsible for halting any ongoing operations related to that Interaction).

IC_EXIT This command is special, causing the trigger to exit the application. Typically, the File menu already has a trigger set to this command.

IC_HELP This command indicates that this GenTrigger brings up help, searching for the proper help context (ATTR_GEN_HELP_CONTEXT) and bringing up the help dialog box with that context. Any contexts below the trigger's containing dialog are ignored, however; i.e. the lowest possible context is the that on the dialog box itself. For more information on Help, see "Help Object Library," Chapter 15 of the Concepts Book.

In most cases, you should not provide a visual moniker for any triggers with an **InteractionCommand**; the specific UI will select an appropriate moniker. (You may supply a moniker if you wish to override the specific UI's selection, however.)

Activating an ATTR_GEN_TRIGGER_INTERACTION_COMMAND trigger will cause the UI to send MSG_GEN_GUP_INTERACTION_COMMAND to the trigger itself. This message will travel up to the appropriate GenInteraction object, where it will be handled.

This activity only occurs if both the trigger's destination and its action message fields are null, however. If these fields are non-null (specifying your own action message and destination), the sending of the **InteractionCommand** will be overridden. You may want to do this to replace the default functionality of a Specific UI-supplied trigger. For complete usage of an **InteractionCommand** trigger within a GenInteraction object, see "GenInteraction," Chapter 7.

5.3.3 Interpreting Double-Clicks

ATTR_GEN_TRIGGER_CUSTOM_DOUBLE_PRESS

Your trigger sends out the message in the *GTI_actionMsg* field and data in the ATTR_GEN_TRIGGER_ACTION_DATA field whenever the user clicks on the trigger, whether that click is a single or a double-click. If you wish your trigger to send out an alternate message if the user double-clicks, use ATTR_GEN_TRIGGER_CUSTOM_DOUBLE_PRESS.

If you set a message for this attribute, the trigger may send out two different messages: one for a single click action and another for a double-click action.

Code Display 5-4 A Trigger with a Double-Click Message

```
@object GenTriggerClass DoubleClickTrigger = {

/* If the user single-clicks on this trigger, MSG_MY_ACTION will be sent along
 * with data1 to the process. If the user double-clicks,
 * MSG_MY_DOUBLE_PRESS_ACTION will be sent instead. */

    GTI_actionMsg = MSG_MY_ACTION;
    GTI_destination = process;
    GI_visMoniker = “Single or Double Click!”;
    ATTR_GEN_TRIGGER_ACTION_DATA = data1;
    ATTR_GEN_CUSTOM_DOUBLE_PRESS = MSG_MY_DOUBLE_PRESS_ACTION;
}
```

5.3.4 Initiating an Action

MSG_GEN_TRIGGER_SEND_ACTION, MSG_GEN_ACTIVATE

In almost all cases, the specific UI initiates the action of the GenTrigger. Once the specific UI determines that a trigger is activated, it will send a MSG_GEN_TRIGGER_SEND_ACTION to the object, which will in turn send the message in *GTI_actionMsg* to the object in *GTI_destination*. In some rare cases, however, you may wish to manually “trigger” the GenTrigger yourself. This can be done by sending the object the MSG_GEN_TRIGGER_SEND_ACTION yourself.

Note that this message, in effect, skips the specific UI and thus any specific functionality defined for activating a trigger will be ignored. In OSF/Motif, for example, a trigger may “blink” (display itself in reverse video) when activated. To perform the default specific UI behavior, send the object a MSG_GEN_ACTIVATE instead, which will perform the default activation for an object and then call MSG_GEN_TRIGGER_SEND_ACTION.

■ MSG_GEN_TRIGGER_SEND_ACTION

```
void    MSG_GEN_TRIGGER_SEND_ACTION(
        Boolean doublePressFlag);
```

This message causes a GenTrigger to send its action message specified in *GTI_actionMsg* to the output specified in *GTI_destination* if the object is

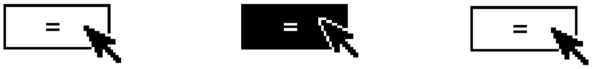


Figure 5-3 *Default GenTrigger activation.*
A GenTrigger in OSF/Motif might "blink" when activated.

GS_USABLE. This message is sent automatically by the UI as part of the default behavior when a GenTrigger is activated by the specific UI. If you wish to simulate a double-click, sending the message included in an object's ATTR_GEN_TRIGGER_CUSTOM_DOUBLE_PRESS, pass this message with a value of *true*. Double-press messages may not pass any data in a trigger's ATTR_GEN_TRIGGER_ACTION_DATA.

Slightly more useful is the **GenClass** message MSG_GEN_ACTIVATE, which activates the trigger at the specific UI's level, performing any default activation behavior (such as making the trigger blink to show that it is being activated).

Source: Unrestricted. Usually the specific UI.

Destination: Any usable GenTrigger object

Parameters: *doublePressFlag* Pass *true* to use the message stored in ATTR_GEN_TRIGGER_CUSTOM_DOUBLE_PRESS rather than message in *GTI_actionMsg*.

Return: Nothing.

Interception: Generally not intercepted.

See Also: MSG_GEN_ACTIVATE

5.3.5 Setting a Trigger As the Default

MSG_GEN_TRIGGER_MAKE_DEFAULT_ACTION

Often, a dialog box or other windowed generic object will have several triggers. You will likely want one of these triggers to be the default action—the default action is activated when the user double-clicks on some item in the dialog or when the user hits Return, typically. (Specific UIs can determine how the activation occurs.)

To set a trigger the default, use HINT_DEFAULT_DEFAULT_ACTION; this only works for the first time the trigger is created, though. To set a trigger to be the default action item dynamically, use the message MSG_GEN_TRIGGER_MAKE_DEFAULT_ACTION.

■ MSG_GEN_TRIGGER_MAKE_DEFAULT_ACTION

```
void      MSG_GEN_TRIGGER_MAKE_DEFAULT_ACTION();
```

This message sets the trigger to be the default action in its windowed parent. It is the message equivalent of HINT_DEFAULT_DEFAULT_ACTION, defined in **GenClass**.

Source: Unrestricted.

Destination: Any GenTrigger object.

Interception: Generally not intercepted.

5.3.6 Manipulating Instance Data

MSG_GEN_TRIGGER_GET_ACTION_MSG,
MSG_GEN_TRIGGER_SET_ACTION_MSG,
MSG_GEN_TRIGGER_GET_DESTINATION,
MSG_GEN_TRIGGER_SET_DESTINATION

A GenTrigger's instance data need not be static. The instance data can be altered dynamically using the following messages of **GenTriggerClass**.

■ MSG_GEN_TRIGGER_GET_ACTION_MSG

```
Message MSG_GEN_TRIGGER_GET_ACTION_MSG();
```

This message retrieves the *GTI_actionMsg* instance data of the trigger.

Source: Unrestricted.

Destination: Any GenTrigger object.

Parameters: None.

Return: Message in *GTI_actionMsg*.

Interception: Generally not intercepted.

■ MSG_GEN_TRIGGER_SET_ACTION_MSG

```
void        MSG_GEN_TRIGGER_SET_ACTION_MSG(
            Message message);
```

This message sets the *GTI_actionMsg* instance field of a GenTrigger to the message passed.

Source: Unrestricted.

Destination: Any GenTrigger object.

Parameters: *message* Message to assign *GTI_actionMsg* to.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_TRIGGER_GET_DESTINATION

```
optr        MSG_GEN_TRIGGER_GET_DESTINATION();
```

This message returns the *GTI_destination* instance data field of the GenTrigger.

Source: Unrestricted.

Destination: Any GenTrigger object.

Parameters: The optr of destination object (in *GTI_destination*).

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_TRIGGER_SET_DESTINATION

```
void        MSG_GEN_TRIGGER_SET_DESTINATION(
            optr dest);
```

This message sets the *GTI_destination* instance field of the GenTrigger to the optr passed.

Source: Unrestricted.

Destination: Any GenTrigger object.

Parameters: *dest* The optr of the new destination.

Return: Nothing.

Interception: Generally not intercepted.

5.3.7 Other Hints

```
HINT_TRIGGER_BRINGS_UP_WINDOW,
ATTR_GEN_TRIGGER_IMMEDIATE_ACTION,
HINT_TRIGGER_DESTRUCTIVE_ACTION
```

Besides the hints discussed in earlier sections and the hints inherited from **GenClass**, **GenTriggerClass** provides two others, as well as one additional attribute:

- HINT_TRIGGER_BRINGS_UP_WINDOW
  Use this hint if your trigger action indirectly brings up a window (such as a GIV_DIALOG GenInteraction or some other interaction window). This hint, depending on the specific UI, may be used to display the button with an ellipsis "…" to symbolize that the trigger leads to some other interaction.

- ATTR_GEN_TRIGGER_IMMEDIATE_ACTION
  Use this attribute if your trigger is in a menu (GIV_POPUP interaction) and its action should take place immediately before other pending UI events. This will prevent default UI events from occurring before the trigger's action. For example, this attribute is used for the tack trigger in the menu box, to prevent the menu from being closed before it is tacked.

- HINT_TRIGGER_DESTRUCTIVE_ACTION
  Use this hint if your action may be potentially destructive. This hint prevents the trigger from being recognized as the default trigger.

For more information on other hints not specific to GenTriggers see "Managing UI Geometry," Chapter 12 of the Concepts Book and hints defined under **GenClass**.

5.4 Customizations

You may customize a GenTrigger using all the basic **GenClass** attributes as shown in "GenClass," Chapter 2. Of special significance are the ways in which you can tailor a moniker to print a graphics string on a button, as in a DOS room icon. The moniker should have as its argument a GString. (See "Graphics Environment," Chapter 23 of the Concepts Book).

Code Display 5-5 Code for a “Graphic Button” GenTrigger

```
@start TriggerGraphicsMonikerResource, notDetachable;

@visMoniker TriggerGraphicsMoniker = {
        size = standard;
        color = color4;
        aspectRatio = normal;
        cachedSize = 95, 70;
        gstring {
                GSBeginString
                GSDrawBitmapAtCP <(EndTriggerGraphicsIcon-
                                        StartTriggerGraphicsIcon)>
                StartTriggerGraphicsIcon label byte
                <...Bitmap here...>
                EndTriggerGraphicsIcon label byte
                GSEndString
        }
}

@end    TriggerGraphicsMonikerResource;

@object GenTriggerClass GraphicTrigger = {
        GI_visMoniker = list
                {
                TriggerGraphicsMoniker
                }
        GTI_actionMsg = MSG_CUSTOM_MESSAGE;
        GTI_destination = process;
        ATTR_GEN_TRIGGER_ACTION_DATA = {DataToPass};
}
```

5.4

◆Objects

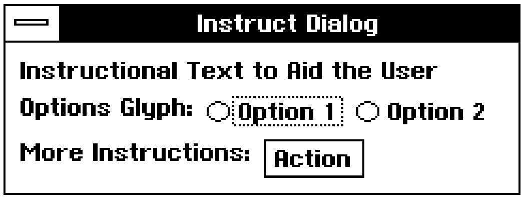
GenGlyph

6

The GenGlyph object allows you to easily display single lines of text without using a more complex GenText object. A GenGlyph can be thought of as a label in which you can display text but which offers no other functionality.

6.1 GenGlyph Features

Frequently throughout your application, you may need to write simple lines of text. In many cases, instructional text is needed to request the user to make a selection. In other cases, text may be needed to alert the user to a certain application condition. The GenGlyph offers a simple, easy to use text display object. Most often, a GenGlyph is used to display instructional text within dialog boxes.


Instruct Dialog

Instructional Text to Aid the User

Options Glyph: ○ Option 1 ○ Option 2

More Instructions: Action

Figure 6-1 *A Collection of GenGlyphs.*
Glyphs are useful to write instructional text. The “Instructional Text...”, “Options Glyph:” and “More Instructions:” text fields are all GenGlyphs.

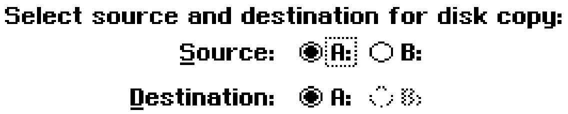
One other generic object, GenText, provides a powerful means to display text with functions such as word-wrapping, font scalability, and an arbitrarily large buffer size. If you do not need such a versatile and powerful text display capability, however, it is wiser and easier to use a GenGlyph. This object only displays text in the default font and style for your application.

6.2 GenGlyph Basics

The GenGlyph object does not, by itself, offer any instance fields or messages. The functionality of a GenGlyph object is included entirely within the instance fields and messages of its superclass, **GenClass**.

Unlike text within a GenText, the text of a GenGlyph is taken solely from the object's *GI_visMoniker*. Therefore, the primary use of a GenGlyph lies in its visual moniker. Any **GenClass** messages dealing with an object's visual moniker will modify the appearance of a GenGlyph.

A GenGlyph does not allow word-wrapping; if your text may be longer than the object's horizontal constraints, either break up the lines of text into smaller, single-line GenGlyphs or use a GenText object.



Select source and destination for disk copy:

Source: ● A: ○ B:

Destination: ● A: ○ B:

Figure 6-2 *Examples of GenGlyphs.*

In this example, the instruction line, along with the "Source:" and "Destination:" text objects are all GenGlyphs.
The radio buttons (A:, B:) are GenItems within GenItemGroups.

Code Display 6-1 Using Hints to Manage GenGlyphs

```
/* This code will duplicate the display shown in Figure 6-2 */

@object GenInteractionClass DiskCopyBox = {
    GI_comp = @DiskCopyHeader, @SourceSelection, @DestinationSelection;
    GII_visibility = GIV_DIALOG;
    HINT_ORIENT_CHILDREN_VERTICALLY;
}

/* The DiskCopyHeader will be the line of instructional text. */
```

```
@object GenGlyphClass DiskCopyHeader = {
    GI_visMoniker = “Select source and destination for disk copy:”;
    HINT_CENTER_MONIKER;/* Centers the moniker horizontally. */
}

/* These objects will form the “Source” line. */

@object GenInteractionClass SourceSelection = {
    GI_comp = @SourceText, @SourceList;
    HINT_ORIENT_CHILDREN_HORIZONTALLY;
    HINT_CENTER_MONIKER;
}

/* This group of objects will form the “Destination” line. */

@object GenInteractionClass DestinationSelection = {
    GI_comp = @DestinationText, @DestinationList;
    HINT_ORIENT_CHILDREN_HORIZONTALLY;
    HINT_CENTER_MONIKER;
}

/* For simplicity, the List objects are not shown. */

@object GenGlyphClass SourceText = {
    GI_visMoniker = “Source:”;
}

@object GenInteractionClass DestinationText = {
    GI_visMoniker = “Destination:”;
}
```

6.3 Modifying a GenGlyph

You may wish at some point to change the text in a GenGlyph. Because a GenGlyph’s text is entirely within its visual moniker, it is a simple matter to use **GenClass** messages to change the moniker dynamically. Use MSG_GEN_GET_VIS_MONIKER to return a GenGlyph’s current moniker; use MSG_GEN_USE_VIS_MONIKER (or MSG_GEN_REPLACE_VIS_MONIKER) to set a new moniker for a GenGlyph. Doing so will ensure that the object is visually rebuilt with the new moniker.

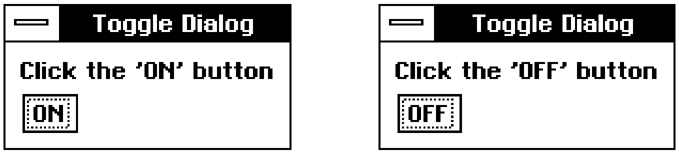


Figure 6-3 *A Toggleable GenGlyph*
This Glyph shows the before and after states of Code Display 6-2.

Code Display 6-2 A Toggle On/Off Switch Dialog Box

```
@object GenInteractionClass MyDialogBox = {
    GI_comp = @DialogText, @DialogButton;
    GII_visibility = GIV_DIALOG;
}

/* Monikers for the Instructions (the GenGlyphs). */

@visMoniker OnTextMoniker = “Click the ‘ON’ button”;
@visMoniker OffTextMoniker = “Click the ‘OFF’ button”;

/* Monikers for the triggers. */

@visMoniker OnButtonMoniker = “ON”;
@visMoniker OffButtonMoniker = “OFF”;

/* The object begins in the “On” state. */

@object GenGlyphClass DialogText = {
    GI_vismoniker = @OnTextMoniker;
}

/* Whenever the button is pressed, MSG_FLIP_THE_SWITCH will change both the glyph
 * and the trigger monikers. */

@object GenTriggerClass DialogButton = {
    GI_visMoniker = @OnButtonMoniker;
    GTI_actionMsg = MSG_FLIP_THE_SWITCH;
    GTI_destination = process;
}

@method MyProcessClass, MSG_FLIP_THE_SWITCH {
    ChunkHandle testMoniker;     /* Stores the temporary moniker. */
    testMoniker = @call DialogButton::MSG_GEN_GET_VIS_MONIKER;

/* If the moniker is “ON”, turn both it and the glyph to the Off monikers.
 * Otherwise (the moniker is “OFF”), turn both it and the glyph to the On
 * monikers. Both visual updates are delayed via the UI queue (and will therefore
 * be updated at the same time rather than separately) to avoid flashing. */

    if (testMoniker == “ON”) {
        @call DialogButton::MSG_GEN_USE_VIS_MONIKER(OptrToChunk(@OffButtonMoniker),
                                                  VUM_DELAYED_VIA_UI_QUEUE);
        @call DialogText::MSG_GEN_USE_VIS_MONIKER(OptrToChunk(@OffTextMoniker),
                                                  VUM_DELAYED_VIA_UI_QUEUE);
    }
    else {
        @call DialogButton::MSG_GEN_USE_VIS_MONIKER(OptrToChunk(@OnButtonMoniker),
                                VUM_DELAYED_VIA_UI_QUEUE);
        @call DialogText::MSG_GEN_USE_VIS_MONIKER(OptrToChunk(@OnTextMoniker),
                                VUM_DELAYED_VIA_UI_QUEUE);
        }

    }
```

6.3

◆Objects

GenInteraction

7

GenInteraction is a high-level object whose purpose is to manage groups of other generic objects. These groups of objects can appear as part of a window, as their own menu or even as a separate dialog box. **GenInteractionClass** also provides the means to interpret actions of its children and adjust the behavior of the Interaction accordingly. Therefore, it is meaningless for an Interaction to appear without children to manage.

GenInteractionClass is a subclass of **GenClass** and therefore inherits all of its functionality. Of special interest for GenInteractions is geometry management with hints provided in **GenClass**. See “GenClass,” Chapter 2 and “Managing UI Geometry,” Chapter 12 of the Concepts Book for more information.

You should be familiar with generic objects before reading this chapter. Please see “The GEOS User Interface,” Chapter 10 of the Concepts Book for an overview of each generic object in the user interface. You should be particularly familiar with the instance data of and messages handled by **GenClass**. (See “GenClass,” Chapter 2.)

7.1 GenInteraction Features

Because of the nature of **GenInteractionClass**, objects of this class may appear in a variety of forms. The specific UI will have some control over the visual presentation of GenInteractions, but your application can usually specify in what manner the Interaction should appear. Your application may also add hints to the specific UI to modify the arrangement of objects within the Interaction.

GenInteraction provides three main visual implementations:

- A sub-group, having no inherent visual representation at all. (The visual implementation depends on the Interaction’s parent.)
- A popup (typically a menu) remaining hidden from the application until activated, and then normally only remaining on-screen for the duration of the selection.

- An independently displayable window (dialog box) coming on-screen whenever activated and going off-screen whenever instructed to.
- Popout dialog boxes which can act as sub-group GenInteractions (for example a menu bar) that can be “popped out” into dialog boxes. Note that popouts are not related to pop-ups.

Specifically, a GenInteraction also provides the following features:

7.1

- organization of the spacial arrangement of other objects.
- the capability to apply properties within an Interaction either immediately (in immediate mode) or at a later time (in delayed mode).
- the display of progress reports on the state of a process or object.
- the ability to initiate commands.
- several default dialog boxes for notification and affirmation responses.
- modal dialog boxes that block other threads of execution until responded to.

7.1.1 Sub-Group Interactions

A sub-group Interaction is the simplest manifestation of a GenInteraction. A sub-group may only appear as part of a larger window. You will most often use sub-group Interactions to arrange and group other objects (the Interaction’s children) within proper places in the UI. This grouping is also sometimes useful to perform geometry actions upon and otherwise manipulate a group of objects as a whole. Using Interactions eliminates the need to provide individual requirements to each child object.

For example, you can use the hint HINT_ORIENT_CHILDREN_VERTICALLY on a single GenInteraction object to display its children in a vertical row. The Interaction’s only function in this case is to provide an object for the hint to act upon. Interactions of this type may contain any number and type of children. Because they are not independently displayable, however, their use becomes limited to the constraints of the larger window. If you need the Interaction’s children to be hidden from view until needed, you should use a popup or dialog Interaction.

◆Objects

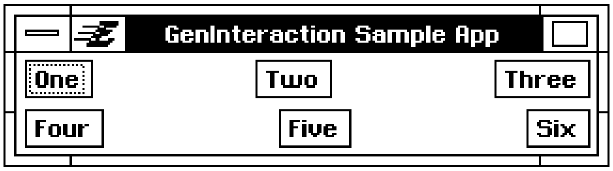


Figure 7-1 *Sub-Group Interactions*
The GenPrimary (GenInteraction Sample App) contains two Sub-Group Interaction children. The first groups the three children One, Two, and Three. The second groups the triggers Four, Five, and Six. The geometry is out of kilter due to different child spacing.

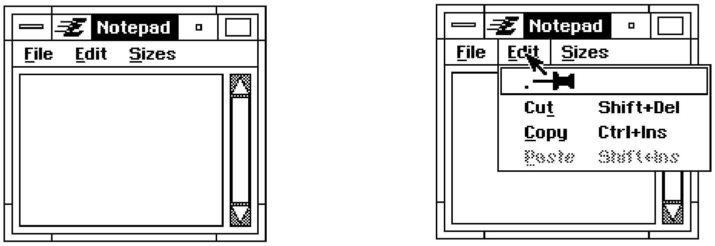
7.1.2 Popup Interactions (Menus)

A popup is an Interaction that you can use to contain commands or selections that should only be displayed when the popup itself is activated. A popup usually conceals its children from view until needed, therefore avoiding cluttering up the UI with dozens of objects at any given time. Popups are most often implemented as menus. Popups usually only remain on-screen until either a selection within the popup is made or the user activates some other object. Menus in OSF/Motif, for example, will only remain on-screen until an object in the menu is selected or until the user clicks somewhere else in the application. (GEOS' implementation of OSF/Motif does allow popups to be pinned, however.)

Popups are not independently displayable; they must appear as part of another window. The popup Interaction's visual moniker usually serves as the only visual cue to the existence of the menu until it is activated. These menu titles usually appear in an appropriate menu bar of the window.

7.1.3 Dialog Boxes

A dialog box is an Interaction that takes the form of an independently displayable window. Typically, a dialog box is a temporary window brought

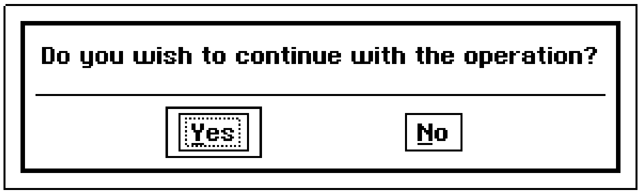


7.1

Figure 7-2 *Popups in a GenPrimary.*

The GenInteraction's visual monikers are the only visual cue to the existence of popups. Once a menu is activated (by clicking on its menu title), the menu appears.

up by the application to request information from the user. Dialogs differ from Primary and Document windows in several ways, however. (For example, most dialogs are not resizable.).



Do you wish to continue with the operation?

Figure 7-3 *A Dialog Box.*

This dialog box prompts the user to answer either yes or no to a question.

A dialog box can be in either of two conditions: modal or non-modal. By default, dialog boxes are non-modal. A modal dialog box is one in which all other input to the application is blocked until the user responds to the dialog. You will usually need modal dialog boxes when an application must gain some further information before proceeding. For example, many of the System Preferences dialog boxes are modal to prevent the user from changing other aspects of the system while these dialogs are up.

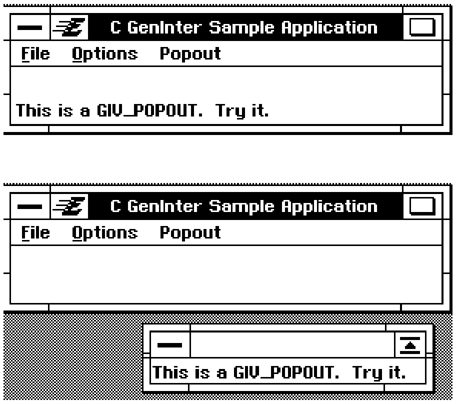
Objects

Dialog boxes come on-screen either through direct user activation or through the application's instigation. By default, a dialog box will usually have an associated trigger object placed within the UI. This trigger's only function is to allow the user to bring the dialog box on-screen. You may remove this default activation trigger if your application code should bring up the dialog box instead.

7.1.4 PopOuts

A popout GenInteraction is really just a grouping object that exists as either a simple sub-group or as a dialog box. Popouts generally begin life as sub-groups but can be "popped out" into dialog boxes that can be repositioned wherever the user wishes them. The dialog boxes can also be popped back into their sub-group arrangement.

In general, the GenInteraction is either popped-out or popped-in by double-clicking within the Interaction bounds.



This is a GIV_POPOUT. Try it.

This is a GIV_POPOUT. Try it.

Figure 7-4 *Popouts*

In the sample application, the popout GenInteraction titled "This is a GIV_POPOUT, try it." is acting as a sub-group below the GenPrimary. When it is double-clicked on, it pops out into the dialog box shown.

7.2 GenInteraction Instance Data

GenInteraction has a set of instance fields; all are listed in Code Display 7-1. Any objects of **GenInteractionClass** or one of its subclasses will contain these instance fields, along with the instance fields of **GenClass**.

7.2 **Code Display 7-1 GenInteraction Instance Fields**

```
@instance GenInteractionType GII_type = GIT_ORGANIZATIONAL;

typedef ByteEnum GenInteractionType;
#define GIT_ORGANIZATIONAL 0
#define GIT_PROPERTIES 1
#define GIT_PROGRESS 2
#define GIT_COMMAND 3
#define GIT_NOTIFICATION 4
#define GIT_AFFIRMATION 5
#define GIT_MULTIPLE_RESPONSE 6

@instance GenInteractionVisibility GII_visibility = GIV_SUB_GROUP;

typedef ByteEnum GenInteractionVisibility;
#define GIV_NO_PREFERENCE 0
#define GIV_POPUP 1
#define GIV_SUB_GROUP 2
#define GIV_CONTROL_GROUP 3
#define GIV_DIALOG 4
#define GIV_POPOUT 5

@instance GenInteractionAttrs GII_attrs = 0;

typedef ByteFlags GenInteractionAttrs;
#define GIA_NOT_USER_INITIATABLE        0x80
#define GIA_INITIATED_VIA_USER_DO_DIALOG 0x40
#define GIA_MODAL                       0x20
#define GIA_SYS_MODAL                   0x10
```

GII_type describes the function of the GenInteraction.

GII_visibility describes the visual implementation of the GenInteraction. Typically, this affects where and how the object and its children are arranged within the UI. A GenInteraction's visibility may also affect the behavior of the children, though this is generally a function of the *GII_type* field.

GII_attrs define attributes that affect how a GenInteraction is initiated, and whether input to other parts of the application (or system) is affected.

Code Display 7-2 GenInteraction Hints

```
/*
 * Hints that affect Properties GenInteractions. These hints are described in the
 * GIT_PROPERTIES GenInteraction section.
 */

@vardata void HINT_INTERACTION_SINGLE_USAGE;
@vardata void HINT_INTERACTION_COMPLEX_PROPERTIES;
@vardata void HINT_INTERACTION_SIMPLE_PROPERTIES;
@vardata void HINT_INTERACTION_RELATED_PROPERTIES;
@vardata void HINT_INTERACTION_UNRELATED_PROPERTIES;
@vardata void HINT_INTERACTION_SLOW_RESPONSE_PROPERTIES;
@vardata void HINT_INTERACTION_FAST_RESPONSE_PROPERTIES;
@vardata void HINT_INTERACTION_REQUIRES_VALIDATION;

/* Hints that affect all types of GenInteractions. */

@vardata void HINT_INTERACTION_FREQUENT_USAGE;
@vardata void HINT_INTERACTION_INFREQUENT_USAGE;
@vardata void HINT_INTERACTION_MAKE_RESIZABLE;
@vardata void HINT_INTERACTION_CANNOT_BE_DEFAULT;
@vardata void HINT_INTERACTION_MODAL;
@vardata void HINT_INTERACTION_NO_DISTURB;
@vardata void HINT_INTERACTION_DEFAULT_ACTION_IS_NAVIGATE_TO_NEXT_FIELD;
@vardata void HINT_CUSTOM_SYS_MENU;
@vardata void HINT_INTERACTION_MAXIMIZABLE;
@vardata void HINT_INTERACTION_POPOUT_HIDDEN_ON_STARTUP;
```

HINT_INTERACTION_FREQUENT_USAGE indicates that the GenInteraction is frequently used, and the specific UI may alter default functionality to reflect this. In most cases, this prevents dialog boxes from being automatically dismissed when the user is done interacting with them.

HINT_INTERACTION_INFREQUENT_USAGE indicates that the GenInteraction is infrequently used. In some specific UIs, this prevents sub-menus from automatically cascading.

HINT_INTERACTION_MAKE_RESIZABLE indicates that this GenInteraction may be resized even if it not normally allowed to do so. This hint usually only applies to dialog boxes.

HINT_INTERACTION_CANNOT_BE_DEFAULT indicates that this GenInteraction should not be activated as a system default. This hint prevents children of this GenInteraction from being activated as system defaults also. You should use this hint to mark groups of objects that may be potentially destructive; the user will be prevented from accidentally activating any objects in the GenInteraction.

HINT_INTERACTION_MODAL indicates that this GenInteraction (a dialog box) should be application modal. This hint should be used in place of GIA_MODAL in cases where modality is not needed for functionality, but is needed to present a cleaner UI to the user. Ideally, you should be able to go back and remove these hints by reworking the UI at some later time.

HINT_INTERACTION_NO_DISTURB indicates that this GenInteraction, if initiated, should be brought on-screen without disturbing the focus of the application. Usually, this hint is placed on notification type dialog boxes (such as new-mail notifications) so that the dialog box doesn't irritate the user by drawing away the focus.

HINT_INTERACTION_DEFAULT_ACTION_IS_NAVIGATE_TO_NEXT_FIELD indicates that the default action for this Interaction group is to navigate to the next field. If something activates the GenInteraction's default action (for example by double-clicking within its confines) objects within the Interaction will not be activated; the focus will travel to the next field.

HINT_CUSTOM_SYS_MENU indicates that this GenInteraction menu is a custom system menu. Depending on the specific UI, the standard system menu will be added as a sub-menu of the custom system menu. This hint takes an integer value that specifies the child position to place the standard system menu (0 being the first position).

HINT_INTERACTION_MAXIMIZABLE indicates that this GenInteraction should be maximizable. This hint also allows the GenInteraction to become restorable to its non-maximizable state.

HINT_INTERACTION_POPOUT_HIDDEN_ON_STARTUP indicates that the GIV_POPOUT GenInteraction should be initially off-screen.

Objects

Code Display 7-3 GenInteraction Optional Attributes

```
/* Optional Attributes. */

@vardata byte ATTR_GEN_INTERACTION_GROUP_TYPE;   /* GenInteractionGroupType */

typedef ByteEnum GenInteractionGroupType;
#define GIGT_FILE_MENU 0
#define GIGT_EDIT_MENU 1
#define GIGT_VIEW_MENU 2
#define GIGT_OPTIONS_MENU 3
#define GIGT_WINDOW_MENU 4
#define GIGT_HELP_MENU 5
#define GIGT_PRINT_GROUP 6

@vardata void ATTR_GEN_INTERACTION_OVERRIDE_INPUT_RESTRICTIONS;
@vardata void ATTR_GEN_INTERACTION_ABIDE_BY_INPUT_RESTRICTIONS;
@vardata void ATTR_GEN_INTERACTION_POPPED_OUT;
@vardata void ATTR_GEN_INTERACTION_POPOUT_NOT_CLOSABLE;
```

ATTR_GEN_INTERACTION_GROUP_TYPE indicates that this GenInteraction (typically a menu) is a special group that the specific UI should be aware of. This optional attribute takes a **GenInteractionGroupType** as its argument. For more information on these standard menus, see “Standard Interactions (Menus)” on page 398.

ATTR_GEN_INTERACTION_OVERRIDE_INPUT_RESTRICTIONS instructs the specific UI to override any input restrictions in place on a modal dialog box. Many dialog types (GIT_PROGRESS, GIT_NOTIFICATION, GIT_AFFIRMATION and GIT_MULTIPLE_RESPONSE) override these input restrictions by default anyway; this hint is for use in cases where this override is not provided.

ATTR_GEN_INTERACTION_ABIDE_BY_INPUT_RESTRICTIONS instructs the specific UI to abide by any input restrictions in place on a modal dialog box. In general, this changes the default behavior for many dialog types (GIT_PROGRESS, GIT_NOTIFICATION, GIT_AFFIRMATION and GIT_MULTIPLE_RESPONSE) which normally override any input restrictions be default.

ATTR_GEN_INTERACTION_POPPED_OUT indicates that a GIV_POPOUT GenInteraction is in its popped-out state. Absence of this optional attribute indicates that the popout is in its popped-in state. This attribute can be set initially, in which case the Popout will appear popped out. This attribute is also set internally by the UI whenever the state of the GIV_POPOUT changes.

ATTR_GEN_INTERACTION_POPOUT_NOT_CLOSABLE indicates that a GIV_POPOUT, when popped out in its dialog box state, will not be closable; you will only be able to pop it back into its sub-group state.

7.2.1 GenInteraction Visibility

```
GII_visibility, MSG_GEN_INTERACTION_GET_VISIBILITY,
MSG_GEN_INTERACTION_SET_VISIBILITY
```

The *GII_visibility* instance field describes in what manner the user interface will display the Interaction. This is important for GenInteractions because of the wide variety of ways they may appear within an application. In most cases, this attribute will not directly affect the functionality of the children of the Interaction but only the visual implementation and specific UI activations of the objects.

Certain *GII_types* (see below) may only be meaningful under certain *GII_visibility* types, however. For example, GIT_NOTIFICATION, GIT_AFFIRMATION, and GIT_MULTIPLE_RESPONSE gain most of their significant functionality within dialog boxes (visibility GIV_DIALOG). In these cases the visibility will affect the functionality of the Interaction.

Each of the following visibilities is an enumeration of type **GenInteractionVisibility**; therefore, you may select one and only one of the following types for your Interaction. By default, a GenInteraction is GIV_SUB_GROUP.

GIV_SUB_GROUP
This specifies that the Interaction serves as a visual grouping of objects within a larger window. This visibility creates an Interaction within the parent window and is therefore not independently displayable. This visibility is the default case.

GIV_POPUP
This specifies that this Interaction is temporary and only appears on-screen for the duration of a single selection. In most specific UIs, the popup will stay up until either an object in the Interaction is activated or until the user clicks somewhere else. Some specific UIs allow a menu to be pinned, however. The

most common manifestation of a popup is a menu. Popups typically appear within appropriate "menu bars" of an application but may be independently displayable under different specific UIs.

GIV_DIALOG This specifies that this Interaction should appear as a dialog box. A dialog box is a temporary window used to display controls or request information from the user. A dialog box can typically contain any number or variety of other generic objects. A dialog box may be either modal or non-modal.

GIV_CONTROL_GROUP
This specifies that this Interaction contains controls and therefore should not appear as a popup Interaction. This prevents the Interaction from disappearing before the user can select an entry in the control group. A control group Interaction may appear as either a sub-group or dialog, depending on the specific UI and its position within the application's generic object tree.

GIV_NO_PREFERENCE
This specifies that there is no visual preference for this Interaction. The specific UI will create the Interaction based on hints, the types of children it contains, and its location in the generic tree.

You may also retrieve or set the visibility of any Interaction at run-time. To retrieve the **GenInteractionVisibility** stored in *GII_visibility*, send the Interaction a MSG_GEN_INTERACTION_GET_VISIBILITY. You can set the visibility of any Interaction by sending it a MSG_GEN_INTERACTION_SET_VISIBILITY. Make sure that any Interaction you set the visibility for is not currently GS_USABLE or an error will result.

■ MSG_GEN_INTERACTION_GET_VISIBILITY

```
byte    MSG_GEN_INTERACTION_GET_VISIBILITY();
```

This message retrieves the current **GenInteractionVisibility** stored in the *GII_visibility* instance field of an Interaction.

Source: Unrestricted.

Destination: Any GenInteraction object.

Parameters: None.

Return: **GenInteractionVisibility** of the Interaction object.

Interception: Generally not intercepted.

■ MSG_GEN_INTERACTION_SET_VISIBILITY

```
void    MSG_GEN_INTERACTION_SET_VISIBILITY(
        byte   visibility);
```

This message sets the *GII_visibility* instance data for the GenInteraction. This message must pass a valid **GenInteractionVisibility** type to the Interaction object. The Interaction must not be GS_USABLE when receiving this message. The new visibility will take effect when the Interaction is next made GS_USABLE.

7.2

Source: Unrestricted.

Destination: Any non-usable GenInteraction object.

Parameters: *visibility* **GenInteractionVisibility** for interaction.

Return: Nothing.

Interception: Generally not intercepted.

Warnings: Make sure that the object sent this message is not GS_USABLE.

7.2.2 Standard Interactions (Menus)

`ATTR_GEN_INTERACTION_GROUP_TYPE, GenInteractionGroupType`

Often you will set up standard menus such as the File menu, the Window menu, the Edit menu, and the View menu. You can set these menus up very simply, with the attribute ATTR_GEN_INTERACTION_GROUP_TYPE.

This attribute specifies the type of menu the GenInteraction is via a value of type **GenInteractionGroupType**. This can be any of the following:

GIT_FILE_MENU
Creates a standard File menu. Typically has a document control and print control as its children.

GIGT_EDIT_MENU
Creates a standard Edit menu with Cut, Copy, Paste, and Undo as appropriate. Typically has a GenEditControl as its child.

GIGT_VIEW_MENU
Creates a standard View menu; typically has a GenView controller as its child.

GIGT_OPTIONS_MENU
Creates a standard Options menu; typically, this menu has application-specific option entries as well as a GenToolControl as its children.

GIGT_WINDOW_MENU
Creates a standard Window menu; typically has a display controller as its child.

GIGT_HELP_MENU
Creates a standard Help menu; typically has the help objects as its children.

GIGT_PRINT_GROUP
Indicates that the GenInteraction is the print group. Typically contains a print controller and other print commands.

Each of the above types will set up the proper GenInteraction attributes and other generic objects to complete the interactions. Thus, by setting up a GenInteraction like this:

```
@object GenInteractionClass MyEditMenu = {
      GI_comp = MyEditControl;
      GII_visibility = GIV_POPUP;
      ATTR_GEN_INTERACTION_GROUP_TYPE =
                  (GIGT_EDIT_MENU);
}
```

you can have a complete Edit menu in your program. (Of course, you must also supply the GenEditControl object that would be MyEditControl in the example.)

7.2.3 GenInteraction Types

```
GII_type, MSG_GEN_INTERACTION_GET_TYPE,
MSG_GEN_INTERACTION_SET_TYPE
```

The *GII_type* instance field describes the contents of the Interaction. This attribute depends on the makeup of the children within the Interaction.

(Note that it would be unusual for an Interaction to appear without children.) In most cases, this will directly affect the functionality of the particular Interaction. Every GenInteraction object should contain children that perform some function within the UI. That function is determined in part by this instance field.

Each of the following types is an enumeration of **GenInteractionType**; therefore, you may select one and only one of the following types for your Interaction. By default, a GenInteraction is GIT_ORGANIZATIONAL.

GIT_ORGANIZATIONAL

This specifies that the Interaction is merely organizational. This allows the Interaction to provide geometry management (and not much else) to its children. Any objects within an organizational Interaction should be able to perform their actions independently. Organizational Interactions may appear in any visual form: sub-groups, menus, or dialog boxes.

GIT_PROPERTIES

This specifies that the Interaction contains properties (attributes) that the user can change. Typically, a properties Interaction will operate in one of two modes: immediate or delayed. If the *GII_visibility* attribute causes this Interaction to appear as a dialog box, the specific UI may create “Apply” and “Reset” triggers. In delayed mode, an apply trigger allows the Interaction to process changes made within the properties Interaction as a group rather than individually; the reset trigger allows the Interaction to reset the properties to their former state. If a popup is GIT_PROPERTIES, it will operate in immediate mode even if hints attempt to override this.

GIT_PROGRESS

This specifies that the Interaction displays a progress report for some operation. For example, a dialog box may appear showing a disk copy operation with the time remaining to its completion. If the *GII_visibility* attribute causes this Interaction to appear as a dialog box, the specific UI may create a “Stop” trigger that halts the operation in progress and the progress reporting. Your application is responsible for stopping the Interaction from communicating its progress report and may dismiss the Interaction if the specific UI desires.

GIT_COMMAND

This specifies that the Interaction contains commands that the application provides. If the *GII_visibility* attribute causes this Interaction to appear as a dialog box, the specific UI may create a "Close" trigger to dismiss the dialog. You will have to supply your own specific command triggers within a GIT_COMMAND dialog box.

GIT_NOTIFICATION

This specifies that the Interaction sends notification of some event. If the *GII_visibility* attribute causes this Interaction to appear as a dialog box, the specific UI may create an "OK" trigger that the user can press to acknowledge the notification.

GIT_AFFIRMATION

This specifies that the Interaction asks for confirmation of an operation. If the *GII_visibility* attribute causes this Interaction to appear as a dialog box, the specific UI may create "Yes" and "No" triggers. Depending upon the Specific UI, these triggers may dismiss the dialog box.

GIT_MULTIPLE_RESPONSE

This specifies that this Interaction may include multiple items that the user can respond to. (Your application should add custom response triggers using the GenTrigger's vardata ATTR_GEN_TRIGGER_INTERACTION_COMMAND.) If you wish these triggers to appear in a dialog box reply bar, use HINT_SEEK_REPLY_BAR.

You may retrieve or set the type of any Interaction at run-time. To retrieve the **GenInteractionType** stored in the *GII_type* instance field, send MSG_GEN_INTERACTION_GET_TYPE. You can set the type of any Interaction by sending it MSG_GEN_INTERACTION_SET_TYPE. Make sure that any Interaction you set the type for is not currently GS_USABLE or an error will result.

■ MSG_GEN_INTERACTION_GET_TYPE

```
byte    MSG_GEN_INTERACTION_GET_TYPE();
```

This message returns the current **GenInteractionType** stored in the *GII_type* instance field for the Interaction.

Source: Unrestricted.

Destination: Any GenInteraction object.

Parameters: None.

Return: **GenInteractionType** of the Interaction.

Interception: Generally not intercepted.

■ MSG_GEN_INTERACTION_SET_TYPE

```
void      MSG_GEN_INTERACTION_SET_TYPE(
          byte   type);
```

7.2

This message sets the *GII_type* instance data for the GenInteraction. This message must pass a valid **GenInteractionType** enumerated type to the Interaction object. The Interaction must not be GS_USABLE when receiving this message; the new type will take effect when the Interaction is next made GS_USABLE.

Source: Unrestricted.

Destination: Any non-usable GenInteraction object.

Parameters: *type* **GenInteractionType** for the Interaction.

Return: Nothing.

Interception: Generally not intercepted.

Warnings: Make sure that the object is not GS_USABLE when sending it this message.

7.2.4 GenInteraction Attributes

```
GII_attrs, MSG_GEN_INTERACTION_GET_ATTRS,
MSG_GEN_INTERACTION_SET_ATTRS
```

The *GII_attrs* instance field describes how the Interaction behaves under various circumstances. These attributes *only* affect the behavior of dialog boxes. If the Interaction given these attributes is not a dialog box, the attributes will have no effect. This field specifies how a GIV_DIALOG Interaction may be initiated and whether input to other parts of the UI is allowed while the Interaction is active. The *GII_attrs* instance field is a bitfield and therefore any combination of these attributes may be set.

None of these attributes is set by default.

GIA_NOT_USER_INITIATABLE

This attribute specifies that the user cannot bring up the dialog directly. Unless this attribute is set, most specific UIs will place a default activation trigger within a proper place in the UI to initiate the dialog box. This attribute prevents this trigger from appearing. To bring up a dialog box set GIA_NOT_USER_INITIATABLE, your application must send the dialog a MSG_GEN_INTERACTION_INITIATE. Usually, these Interactions should be direct children of either a GenPrimary or GenApplication.

GIA_INITIATED_VIA_USER_DO_DIALOG

This attribute specifies that the application brings up this Interaction through **UserDoDialog()**. You should use this routine in special cases where you need the user to respond to a dialog box before continuing with your application thread. (This routine blocks the calling thread until the user responds to the dialog box.) Any dialog boxes marked with this attribute should *not* be initiated with MSG_GEN_INTERACTION_INITIATE.

GIA_MODAL

This attribute specifies that while this dialog is up, all other input to the application should be ignored. This attribute will create an application modal dialog box.

GIA_SYS_MODAL

This attribute specifies that while this dialog is up, all other input to the system (including other applications) should be ignored. This attribute will create a system modal dialog box.

You may retrieve or set the *GII_attrs* of any Interaction at run-time. To retrieve the **GenInteractionAttrs**, send the Interaction a MSG_GEN_INTERACTION_GET_ATTRS. You can set the **GenInteractionAttrs** of any Interaction by sending it a MSG_GEN_INTERACTION_SET_ATTRS. Make sure that any Interaction you set these attributes for is not currently GS_USABLE or an error will result.

If your application contains several dialog boxes sitting directly under a GenPrimary (typically with their GIA_NOT_USER_INITIATABLE attribute), the application may take longer to start up. This happens because the system will have to process each of the dialogs individually and determine at that point that a trigger should not be created for them.

You may instead wish to create an organizational Interaction to manage these non-user initiatable Interactions. To do this, create a single GIT_ORGANIZATIONAL GIA_NOT_USER_INITIATABLE dialog box under the GenPrimary. All "real" dialogs should then be placed as children of this "holding" Interaction. When the application starts up, it only needs to process this one Interaction.

Grouping dialogs in this way also allows easier initiation and dismissal of the dialogs all at once. Simply encapsulate MSG_GEN_INTERACTION_INITIATE or MSG_GEN_GUP_INTERACTION_COMMAND with IC_DISMISS and use MSG_GEN_SEND_TO_CHILDREN on the one Interaction.

■ MSG_GEN_INTERACTION_GET_ATTRS

```
byte      MSG_GEN_INTERACTION_GET_ATTRS();
```

This message retrieves the current *GII_attrs* instance field from the Interaction sent the message.

Source: Unrestricted.

Destination: Any GenInteraction object.

Parameters: None.

Return: **GenInteractionAttrs** of the Interaction.

Interception: Generally not intercepted.

■ MSG_GEN_INTERACTION_SET_ATTRS

```
void      MSG_GEN_INTERACTION_SET_ATTRS(
          byte   setAttrs,
          byte   clearAttrs);
```

This message sets the *GII_attrs* instance data for the GenInteraction sent the message. The Interaction must not be GS_USABLE when receiving this message. The new attributes will take effect when the Interaction is next made GS_USABLE.

Source: Unrestricted.

Destination: Any non-usable GenInteraction object.

Parameters:

setAttrs	**GenInteractionAttrs** to set.
clearAttrs	**GenInteractionAttrs** to clear.

Return: Nothing.

◆ Objects

Interception: Generally not intercepted.

Warnings: Make sure that the object is not GS_USABLE when sending it this message.

7.3 GenInteraction Usage

You will probably need to create GenInteractions at various points within your application. The GenInteraction object is one of the most flexible generic objects within GEOS. Not only can an Interaction operate as a menu, dialog, or sub-group, but there also exist several pre-defined dialog boxes with default triggers for your use. These dialogs will make writing your application easier and faster. The simplest way to see the wide variety of options available to you is to examine the different types of Interactions available with different instance data values.

7.3.1 Visibilities

When first creating an Interaction, you must decide in what form to visually present your communication between the application and the user. This concern may be somewhat cosmetic in form but may also affect the functionality of the Interaction itself. For instance, more than one **GenInteractionType** will support different actions with different visibility types.

7.3.1.1 GenInteraction Sub-Groups (GIV_SUB_GROUP)

The simplest GenInteraction is one used to arrange the geometry of other objects. To do this, a GenInteraction should be GIV_SUB_GROUP (the default).

You can use hints provided in **GenClass** to arrange the appearance of objects within the GenInteraction. Interactions usually provide the greatest flexibility for geometry manipulation. For full information, see “Managing UI Geometry,” Chapter 12 of the Concepts Book.

Note that a GIV_SUB_GROUP Interaction may only appear within a parent window as it is not itself independently displayable.

Code Display 7-4 Arranging a Group of Children

```
/* This Interaction creates a typical number keypad arrangement. */

@object GenInteractionClass MyKeyPad = {
        /* The Interaction by default is GIV_SUB_GROUP and GIT_ORGANIZATIONAL,
         * so there is no need to set these types. */
    GI_comp = Row789, Row456, Row123;
        /* The Interaction’s three children will be arranged vertically. The
         * UI will also draw a border around the Interaction itself. */
    HINT_ORIENT_CHILDREN_VERTICALLY;
    HINT_DRAW_IN_BOX;
}

@object GenInteractionClass Row789= {
    GI_comp = Key7, Key8, Key9;
        /* Each row is then arranged horizontally to gain three rows
         * of three items each. */
    HINT_ORIENT_CHILDREN_HORIZONTALLY;
}

@object GenInteractionClass Row456= {
    GI_comp = Key4, Key5, Key6;
    HINT_ORIENT_CHILDREN_HORIZONTALLY;
}

@object GenInteractionClass Row123= {
    GI_comp = Key1, Key2, Key3;
    HINT_ORIENT_CHILDREN_HORIZONTALLY;
}

/* Each of the child ‘keys’ will be a GenTrigger with appropriate visual
 * moniker. (For simplicity, only one trigger is shown here.) */

@object GenTriggerClass Key9 = {
    GI_visMoniker = “9”;
}
```

Many hints from **GenClass** may affect the appearance of GIV_SUB_GROUP Interactions. See “Managing UI Geometry,” Chapter 12 of the Concepts Book for details.

Objects

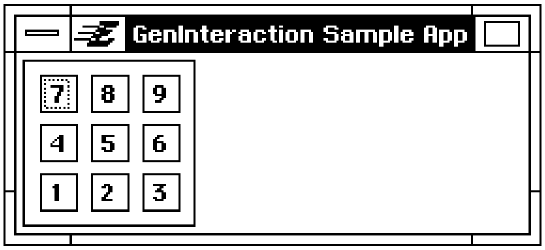


Figure 7-5 *A Keypad example*
The main GenInteraction sub-group is drawn within a box. The smaller sub-groups (the horizontal rows) each contain three trigger children.

Of special interest to sub-group Interactions is HINT_DRAW_IN_BOX. This hint will enclose the associated Interaction within a line border. You should use this hint when you wish to distinguish a collection of objects within an Interaction from some other objects.

7.3.1.2 GenInteractions as Menus (GIV_POPUP)

Most specific UIs implement GIV_POPUP as a menu. Menus are interactions with options that only appear when the menu is activated by the user. Popups make it possible for a user to browse and choose among a variety of commands without cluttering up the UI with dozens of objects. A popup may appear at almost any location within the UI. If a popup is placed as a direct child of a GenPrimary, it may appear in an appropriate menu bar location across the top of the window. (See Figure 7-6.)

A popup is most often one of GIT_ORGANIZATIONAL, GIT_PROPERTIES, or GIT_COMMAND. (See “GenInteraction Types” on page 399.) What type of popup you select depends on the type of children the Interaction contains. For example, an application may implement a font menu as a GIT_PROPERTIES popup. A GIT_COMMAND popup might issue commands relating to the appearance of text within a text editor. Depending on the **GenInteractionType**, a popup may appear in forms other than a menu. (The specific UI has final say over the appearance of a popup.)

A GIV_POPUP Interaction must contain a group of children. If the Interaction appears as a menu, these children will become menu items. Most often, these children will be GenTriggers or GenItems within GenItemGroups.

Code Display 7-5 Using a GenInteraction to Create a Menu

```
/* This example is taken from the Hello World sample application. */

@object GenInteractionClass HelloMenu = {
    GI_visMoniker = ’M’, “Menu”;         /* The moniker of the menu is used in
                                          * the primary window’s menu bar (if the
                                          * specific UI employs a menu bar). */
    GI_comp = HelloColorBox;             /* The only child of the menu (the only
                                          * item in the menu) is the dialog box. */
    GII_visibility = GIV_POPUP;          /* This attribute designates the
                                          * interaction as a menu or a sub-menu. */
}

@object GenInteractionClass HelloColorBox = {
    GI_comp = HelloBlueTrigger, HelloGoldTrigger;
    GI_visMoniker = ’C’, “Color”;
    GII_visibility = GIV_DIALOG;
}

@object GenTriggerClass HelloBlueTrigger = {
    GI_visMoniker = ’B’, “Blue”; /* The ’B’ indicates the keyboard navigation
                                  * character for this trigger. */
    GTI_destination = process;   /* Send the message to the Process object. */
    GTI_actionMsg = MSG_HELLO_CHANGE_TO_BLUE;    /* Send this message. */
}

@object GenTriggerClass HelloGoldTrigger = {
    GI_visMoniker = ’G’, “Gold”; /* The ’G’ indicates the keyboard navigation
                                  * character for this trigger. */
    GTI_destination = process;   /* Send the message to the Process object. */
    GTI_actionMsg = MSG_HELLO_CHANGE_TO_GOLD;    /* Send this message. */
}
```

If a child of the GIV_POPUP Interaction is another Interaction, the specific UI might change the visual implementation of that child. What change takes place depends on the specific UI. In OSF/Motif, for example, placing a sub-group Interaction within a popup Interaction causes the UI to draw lines

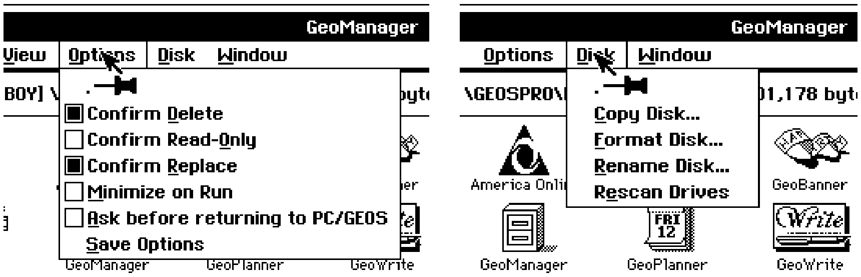


Figure 7-6 *GenInteractions as Popups (Menus)*

Menus can have a variety of children. Here, the "Options" menu consists of a GenBooleanGroup containing five GenBoolean children and a single GenTrigger. The "Disk" menu consists of four GenTriggers.

separating the sub-group from the other children in the popup. (You may prevent this line by adding HINT_SAME_CATEGORY_AS_PARENT.)

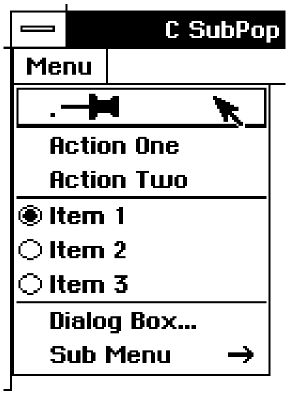


Figure 7-7 *Interactions Within Popups*

This popup (Menu) contains a sub-group interaction (the item group) which is separated from the other menu items by a solid line. The menu items for the dialog box and the sub-menu contain indicators (an ellipsis and an arrow) to show that these menu items lead to other objects.

Placing a popup interaction within another popup is a special case and will create nested menus under most specific UIs. To do this, merely create a GIV_POPUP Interaction as a child of another GIV_POPUP Interaction. The specific UI will automatically create the second menu as a sub-menu of the first. The item in the menu bringing up the sub-menu may be labeled with an arrow under certain specific UIs. (See Figure 7-8.) Menus may be nested to an arbitrary number of levels.

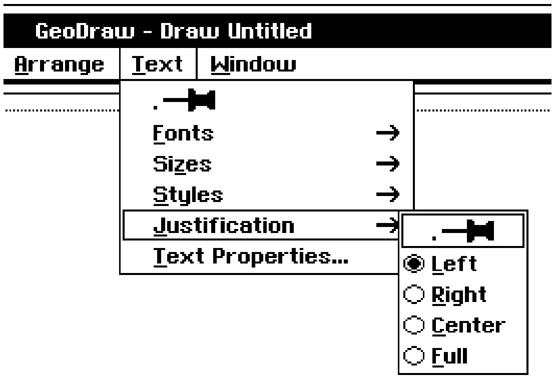


Figure 7-8 *GenInteraction with Nested Menus*

Both the Text and Justification GenInteractions are set GIV_POPUP. Note that OSF/Motif supplies an arrow leading to the sub-menu.

Code Display 7-6 Nested Menus

```
@object GenInteractionClass TextMenu = {
    GI_comp = JustificationSubMenu;
    GI_visMoniker = ‘T’, “Text”;
    GII_visibility = GIV_POPUP;
}

@object GenInteractionClass JustificationMenu = {
    GI_comp = JustificationList;
    GI_visMoniker = ‘J’, “Justification”;
    GII_visibility = GIV_POPUP;
}
```

```
@object GenItemGroupClass JustificationList = {
    GI_comp = LeftEntry, RightEntry, CenterEntry, FullEntry;
        /* These children are not shown. */
}
```

List Objects (GenItemGroups and GenBooleanGroups) within a GIV_POPUP Interaction may be used to show object properties in a menu. The menu should be GIT_PROPERTIES in this case. Depending on the Specific UI, a properties menu may be implemented in a different fashion from other menus. (In OSF/Motif, there is no visual difference between a properties menu and other menus.) Other types have no effect on the behavior or appearance of GIV_POPUPs.

7.3.1.3 GenInteractions as Dialog Boxes (GIV_DIALOG)

The GIV_DIALOG visibility will instruct the specific UI to build the Interaction as a dialog box. A dialog box is an independently-displayable interface element used to display information or other UI objects. Therefore, it is similar to a window although there are some differences. (Most dialog boxes, for example, are not resizable whereas most windows are.)

Dialog boxes also enhance the functionality of other **GenInteractionType**s. Many *GII_types* are only significant as dialog boxes. For example, a GIT_NOTIFICATION Interaction will only appear with an "OK" trigger if it is also a dialog box. A GIT_PROPERTIES Interaction may contain properties within any visibility, but only if built as a dialog can it implement delayed mode. (If a GIT_PROPERTIES Interaction is built as a menu or sub-group, it will always operate in immediate mode.) Delayed mode allows the user to change the properties of an object and later apply those changes all at once. Within a dialog box, an "Apply" trigger will make these changes.

Most specific UIs will build a dialog box with an activation trigger present in the UI that brings up the dialog box. These dialog boxes are known as user initiatable. There are ways to override this behavior if you need a non-user initiatable dialog box or a dialog box that blocks the calling thread until answered (forcing the user to respond to the dialog box immediately). See "Modality for Dialogs" on page 439.

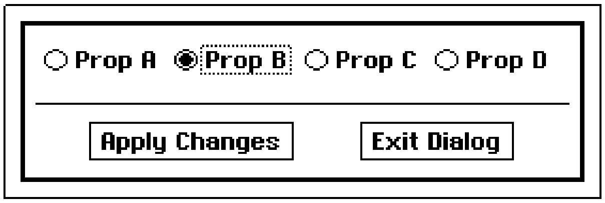


Figure 7-9 *Properties Dialog in Delayed Mode*
This dialog box operates in delayed mode because it supplies a trigger to apply changes made.

By default, dialog boxes are not resizable. They will not contain any resizing mechanism. HINT_INTERACTION_MAKE_RESIZABLE may be used on such a non-modal dialog box to make that dialog resizable. In OSF/Motif, the Specific UI will supply such a dialog with a means to alter its shape.

HINT_INTERACTION_MAKE_RESIZABLE is only meaningful for GIV_DIALOG Interactions. By default, dialog boxes cannot be resized; their dimensions are solely dependent on the number and type of children the dialog contains. Any dialog boxes created with this hint may contain a resizing mechanism to alter their visible dimensions.

7.3.1.4 GenInteractions as Popouts (GIV_POPOUTS)

MSG_GEN_INTERACTION_POP_IN, MSG_GEN_INTERACTION_POP_OUT, MSG_GEN_INTERACTION_TOGGLE_POPOUT

Popouts are special GenInteractions that may act as either sub-group Interactions and as dialog boxes, depending on what state they are in. Usually, popouts act as normal sub-group Interactions until the user or the application “pops” them out into a dialog box. Popouts are normally “popped out” into their dialog box state by double-clicking within the confines of the GenInteraction. This popout state is reflected with either the presence or absence of the ATTR_GEN_INTERACTION_POPPED_OUT vardata entry.

You can pop out a GIV_POPOUT GenInteraction in its sub-group state by sending it MSG_GEN_INTERACTION_POP_OUT. Similarly, you can pop a GIV_POPOUT back into its sub-group state from the dialog box state by sending it MSG_GEN_INTERACTION_POP_IN.

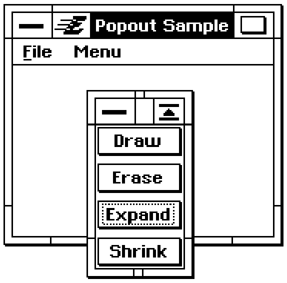


Figure 7-10 *A "popped out" Popout*
The dialog box of the GIV_POPOUT in its popped-out state provides both a default close trigger (on the left hand side of the dialog title bar) and a "pop-in" button, on the right-hand side of the title bar).

MSG_GEN_INTERACTION_TOGGLE_POPOUT toggles a popout from whatever state it currently is into its other state.

By default, popouts in their dialog box state contain a special trigger to pop them back into their sub-group state. They may also be popped back in by double-clicking within the popped-out GenInteraction. Popouts also normally provide a close trigger; this trigger dismisses the dialog box but does not return the popout to its sub-group state. You should provide a trigger somewhere within your UI that sends MSG_GEN_INTERACTION_POP_IN or MSG_GEN_INTERACTION_POP_OUT to the popout, so that the user can bring any popout they dismiss back on-screen. Alternatively, you can add ATTR_GEN_INTERACTION_POPOUT_NOT_CLOSABLE on the Popout to prevent the creation of a "Close" trigger.

7.3.2 Types

Possibly more important than how an Interaction is visually displayed is what role that Interaction should play within your UI. That role depends in part on what children the Interaction contains. What the Interaction does with those children is determined by the **GenInteractionType** of that object. Interactions of different visibility may function in essentially the same manner, but Interactions of different types may differ greatly in their functionality.

For example, if an Interaction contains a GenItemGroup with three GenItem children, it will not affect the basic functionality if that Interaction is built as a dialog, popup, or sub-group. Each item will still perform its default function (such as sending a message). The *GII_type* of Interaction might affect that functionality, however, by forcing the Items to send out their messages as a group or individually (for example in a GIT_PROPERTIES Interaction).

You should pick the **GenInteractionType** based on what actions you wish to perform on the children you select for the Interaction. By default, an Interaction is of type GIT_ORGANIZATIONAL. In some cases, the functionality you need will reside entirely with the children; in those cases, a GIT_ORGANIZATIONAL will most likely suffice.

Certain **GenInteractionType** Interactions, if built as GIV_DIALOG, may create a reply bar with *standard response triggers*. A reply bar is a special area of a dialog Interaction to place commands for user responses. Standard response triggers perform additional actions for dialog Interactions, apart from any functionality your application provides. You do not need to add these triggers yourself; they are added by the Specific UI.

Standard response triggers perform one of several pre-defined **InteractionCommand** commands. Depending on the particular type of Interaction, each command may perform different actions. An **InteractionCommand** is a special data type that the Specific UI recognizes. The Specific UI decides what action to take upon receiving this command; depending on the value of the **InteractionCommand**, the Specific UI may perform several different actions.

For example, the GIT_NOTIFICATION type, if built as a dialog box, may create an “OK” trigger automatically. The trigger will become one of the dialog box’s children. This trigger sends out the **InteractionCommand** IC_OK when activated. The Specific UI will also decide how to implement that IC_OK message. (In OSF/Motif, for example, IC_OK will dismiss the dialog box as one of its actions.)

You can create your own response triggers within an Interaction. You can also replace any standard response trigger by supplying one of your own. For complete information on how **InteractionCommand**s and response triggers work, see “Interaction Commands” on page 450.

7.3.2.1 Organizational Interactions

An organizational interaction is the simplest type of interaction. It is also the default **GenInteractionType**. An organizational Interaction only functions to contain objects—without indicating any meaning to the makeup of those objects. Because of this, a GIT_ORGANIZATIONAL Interaction should not interpret any **InteractionCommand** types. See “Interaction Commands” on page 450.

GIT_ORGANIZATIONAL Interactions are only used, therefore, to group their children; there is never any implied meaning to the children they contain. The Interaction may have a certain visual form defined in its **GenInteractionVisibility** field, but presumably the Interaction should perform its activity under any visible arrangement.

You will most often need an organizational Interaction to arrange objects within another window, whether that “window” is a GenPrimary or a GIV_DIALOG GenInteraction. For example, you may need organizational Interactions solely to arrange a group of generic children in a horizontal or vertical row. There are several hints in **GenClass** that may affect the visual implementation of an organizational Interaction. See “Managing UI Geometry,” Chapter 12 of the Concepts Book for full details.

Organizational Interactions may appear as sub-groups, menus, or dialog boxes; each of its items should already contain all the functionality needed. For example, an edit menu (GIV_POPUP Interaction) with three triggers (Cut, Copy, and Paste) could be organizational if each of those triggers is capable of performing its action purely through the sending of a message (a function of **GenTriggerClass**).

No standard response triggers are provided with a GIT_ORGANIZATIONAL Interaction, even if it is built as a dialog box, though you may supply your own. See “Standard Response Triggers” on page 455.

Code Display 7-7 Organizational Interactions

```
/* This properties dialog box will group a collection of GIT_ORGANIZATIONAL
 * Interactions. */

@object GenInteractionClass MyMasterInteraction = {
    GI_comp = FirstOrganizational, SecondOrganizational;
    GII_visibility = GIV_DIALOG;
    GII_type = GIT_PROPERTIES;
}

@object GenInteractionClass FirstOrganizational = {
/* GenInteractions are GIT_ORGANIZATIONAL by default. */
    GI_comp = One, Two, Three;
}

@object GenInteractionClass SecondOrganizational = {
    GI_comp = Four, Five, Six;
}

@object GenTriggerClass One = {
/* This trigger, after sending out MY_SPECIAL_MESSAGE to the process, will send a
 * MSG_GEN_GUP_INTERACTION_COMMAND with IC_INTERACTION_COMPLETE to the first
 * non-GIT_ORGANIZATIONAL Interaction it encounters up the tree
 * (MyMasterInteraction). This behavior is due to the GenAttribute
 * GA_SIGNAL_INTERACTION_COMPLETE. (See GenClass.) */

    GTI_actionMsg = MY_SPECIAL_MESSAGE;
    GTI_destination = process;
    GI_attrs = GA_SIGNAL_INTERACTION_COMPLETE;
}
```

7.3.2.2 Properties Interactions

GIT_PROPERTIES Interactions allow the Interaction to display and set attributes of a specific selected object (the “current selection”). In many cases, this current selection will be the target of the application. (See “Input,” Chapter 11 of the Concepts Book.) Users can change the UI gadgets in the properties group to change the attributes of the selected object. When the user selects a different object, the UI gadgetry changes to reflect the attributes of the new selection.

For example, a properties dialog may alter text properties within a text editor. If some text is selected, changing properties in the text dialog will change the appearance of the text.

Properties Interactions may appear as sub-groups, popups, or dialog boxes. If a properties Interaction takes the form of a dialog box, the Specific UI may create response triggers with the **InteractionCommand** types IC_APPLY, IC_RESET and IC_DISMISS. These triggers will either apply changes made in the properties group, reset those changes to their initial state, or close the dialog, respectively. In OSF/Motif these triggers are labeled “Apply” or “OK,” “Reset,” and “Close” or “Cancel.”

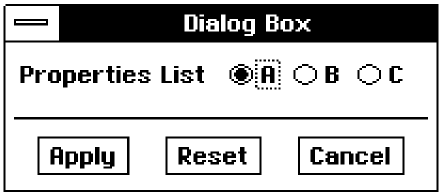


Figure 7-11 *A Properties Dialog Box.*
The Specific UI supplies the “Apply” trigger.

Code Display 7-8 GenInteraction as Properties Dialog Box

```
@object GenInteractionClass MyInteraction = {
    GI_visMoniker = “Dialog Box”;
    GI_comp = PropertiesList;
    GII_visibility = GIV_DIALOG;
    GII_type = GIT_PROPERTIES;
}

/* The GenItemGroup below is a list of three properties. */

@object GenItemGroupClass PropertiesList = {
    GI_visMoniker = “Properties List”;
    GI_comp = PropOne, PropTwo, PropThree;
    GIGI_behaviorType = GIGBT_NON_EXCLUSIVE;
}
```

```
@object GenItemClass PropOne = {
    GI_visMoniker = “A”;
}

@object GenItemClass PropTwo = {
    GI_visMoniker = “B”;
}

@object GenItemClass PropThree = {
    GI_visMoniker = “C”;
}
```

There are two ways that an application can deal with UI gadgetry in a property dialog. Either the application can work completely off the state change notification messages sent by the UI gadgetry when the properties are changed, or the application may need to do some extra work after the properties are changed. This extra work can include checking for valid settings in the UI gadgetry, querying the UI gadgetry for their states, or dealing with the UI gadgetry as a group rather than individually (i.e. fetching each of their states and using them all at once to make a single change instead of making separate changes for each state).

Delayed and Immediate Mode

Properties dialogs work in one of two modes: immediate and delayed. In immediate mode, the UI gadgetry within a properties box reflects the actual state of those objects. Changing selections within the properties Interaction changes the state of those objects immediately. For example, if a “Fonts” dialog box containing a list of font selections is in immediate mode, selecting each separate font will cause the selected text to immediately display itself in the selected font.

In delayed mode, the UI gadgetry within the properties box reflects a separate “user” state rather than the actual state. The user is free to change settings within the properties Interaction without causing such changes to be made on the object immediately. The UI does this by storing two states for the UI gadgetry: the actual state (which reflects the current state of the objects) and a user state (which reflects the state specified in the dialog box). The user can apply changes made within a delayed mode properties Interaction with use of an “Apply” mechanism. Usually, the Specific UI will provide an “Apply” trigger for such a case.

If either the Specific UI or hints determine that the properties dialog box should be run in delayed mode, an apply trigger will be supplied. In OSF/Motif, an “Apply” trigger will be placed in a reply bar along with a “Close” trigger that dismisses the property box.

Properties Hints

7.3

The GIT_PROPERTIES type and GIV_DIALOG visibility indicate to the Specific UI to build a property dialog box. The Specific UI will also determine whether this box should run in immediate or delayed mode. Because the application works completely through its UI gadgets within the dialog, it will support either mode. However, there are good reasons for wanting the properties to work in one mode or another. You can supply additional hints for this purpose.

These hints only affect the behavior of properties dialog boxes (GIV_DIALOG). Therefore, if an Interaction is a GIV_POPUP or GIV_SUB_GROUP, these hints will have no effect. If the Interaction is a GIV_CONTROL_GROUP or GIV_NO_PREFERENCE, these hints may force the object to appear as a dialog box.

HINT_INTERACTION_COMPLEX_PROPERTIES indicates that the properties dialog box contains complex properties and therefore a means to reset the properties to their initial state should be supplied if available. In OSF/Motif, for example, a “Reset” response trigger is supplied.

HINT_INTERACTION_SIMPLE_PROPERTIES indicates that the properties dialog box contains simple properties and therefore a means to reset the properties to their initial state is probably unneeded.

HINT_INTERACTION_RELATED_PROPERTIES indicates that the properties dialog contains properties that are closely related. Therefore, the specific UI should run the objects as a group in delayed mode and provide an “Apply” trigger. This might be useful if you set several graphics area properties (such as color, texture, and dithering) and only wish to send out the changes after the final selection has been made for all the properties. (See Figure 7-12.)

HINT_INTERACTION_UNRELATED_PROPERTIES indicates that the properties dialog contains unrelated properties that may be set individually.

Therefore, the Interaction can be run in immediate mode and no “Apply” trigger is needed.

HINT_INTERACTION_SLOW_RESPONSE_PROPERTIES indicates that the properties dialog contains properties that may take a long time to be reflected in the state of the object. The specific UI may create an “Apply” trigger operating in delayed mode so that the user can set these attributes all at once.

HINT_INTERACTION_FAST_RESPONSE_PROPERTIES indicates that this properties dialog contains properties in which changes can be quickly reflected. The Interaction will be run in immediate mode without any “Apply” and “Reset” triggers.

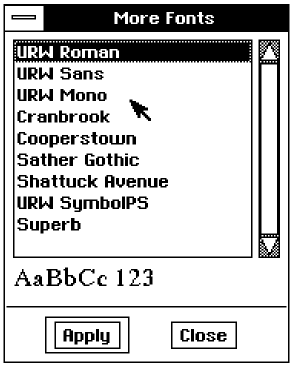


Figure 7-12 *A Properties Dialog*
This “MoreFonts” dialog has HINT_INTERACTION_RELATED_PROPERTIES. OSF/Motif supplies both an “Apply” and a “Close” trigger.

Modifying Triggers for Custom Properties

```
HINT_INTERACTION_REQUIRES_VALIDATION
```

If an application needs to do additional work on a properties dialog box, such as validation of the attributes, you should define a “custom” properties dialog box. You can do this by replacing the default “Apply” and “Reset” triggers with ones of your own design.

HINT_INTERACTION_REQUIRES_VALIDATION indicates that this properties dialog should prompt the user for verification before implementing changes to the properties group. The Interaction will thus be implemented in delayed mode while also allowing a custom “Apply” trigger. The trigger action should be handled in the normal fashion to add any additional functionality to validate before proceeding with the apply.

You can modify the standard IC_APPLY trigger by adding your own apply trigger with ATTR_GEN_TRIGGER_INTERACTION_COMMAND set to IC_APPLY. This trigger will replace the default IC_APPLY trigger that the Specific UI would have supplied otherwise. HINT_SEEK_REPLY_BAR places the trigger in the property dialog’s reply bar. In the GenInteraction itself, you might want to include HINT_INTERACTION_REQUIRES_VALIDATION or another more appropriate hint to ensure that the properties dialog will be working in delayed mode.

An application-supplied trigger allows the application to define whatever action output and method it desires. When the user activates this trigger, the action message will be sent out to the action output (as usual for GenTriggers); the default IC_APPLY functionality will not be performed. In the handler for this custom message, the application can do whatever is necessary with the properties dialog (such as validation of the attributes).

When finished, the application can either manually fetch the state of the attributes or send MSG_GEN_GUP_INTERACTION_COMMAND with IC_APPLY to the GenInteraction. (This is the default behavior of an IC_APPLY **InteractionCommand**.) You may choose not to send out this message if you wish to replace instead of supplement the default behavior.

Finally, you can send MSG_GEN_GUP_INTERACTION_COMMAND with IC_INTERACTION_COMPLETE to notify the GenInteraction that the user has finished a single usage of the Interaction. (You may also mark such a trigger with GA_SIGNAL_INTERACTION_COMPLETE.) Upon this notification, the specific UI will decide if the GenInteraction should be dismissed. Instead, you may send IC_DISMISS to unconditionally dismiss the Interaction.

Code Display 7-9 Replacing the Default IC_APPLY Trigger

```
@object GenInteraction LineProperties = {
    GI_visMoniker = “Line Properties”;
    GI_comp = LineWidth, LineStyle, LinePropertiesApply;
    GII_type = GIT_PROPERTIES;
    GII_visibility = GIV_DIALOG;
    HINT_INTERACTION_REQUIRES_VALIDATION;        /* Forces delayed mode. */
}

@object GenItemGroupClass LineWidth = {
    /* ... */
}

@object GenItemGroupClass LineStyle = {
    /* ... */
}

@object GenTriggerClass LinePropertiesApply = {
    GTI_destination = process;
    GTI_actionMsg = MSG_MY_PROCESS_PROPERTIES_APPLY;
        /* Setting GA_SIGNAL_INTERACTION_COMPLETE will dismiss the dialog
         * after applying the properties. */
    GI_attrs = @default | GA_SIGNAL_INTERACTION_COMPLETE;
        /* Setting this attribute to IC_APPLY tells the specific UI to replace
        * the standard response trigger with this one. */

    ATTR_GEN_TRIGGER_INTERACTION_COMMAND = { IC_APPLY };
    HINT_SEEK_REPLY_BAR;
}
```

The moniker for an application-supplied apply trigger may be set by the application or (preferably) left blank so that the Specific UI can provide an appropriate moniker. The Specific UI uses the IC_APPLY data in the ATTR_GEN_TRIGGER_INTERACTION_COMMAND vardata and any properties hints to determine what moniker to use. (In OSF/Motif this is either “Apply” or “OK.”)

If the application wants to alter a moniker but does not need to do any other work, the trigger’s output fields should be left blank. Because the trigger has ATTR_GEN_TRIGGER_INTERACTION_COMMAND, if the output fields are null, the trigger will send MSG_GEN_GUP_INTERACTION_COMMAND with the ATTR_GEN_TRIGGER_INTERACTION_COMMAND data to itself. This message will travel up to the appropriate properties GenInteraction where it will be handled, as usual. If doing this, you should make sure to mark any replacement triggers GA_SIGNAL_INTERACTION_COMPLETE so that the Specific UI can determine whether to dismiss the dialog or not when the trigger is activated. (See Code Display 7-10.)

Code Display 7-10 Replacing the Default IC_APPLY moniker

```
@object GenTriggerClass LinePropertiesApply = {
    GI_visMoniker = “Set Line Properties”;
    GI_attrs = @default | GA_SIGNAL_INTERACTION_COMPLETE;
    ATTR_GEN_TRIGGER_INTERACTION_COMMAND = { IC_APPLY };
    HINT_SEEK_REPLY_BAR;
}
```

An alternate way to supplement an apply trigger’s default behavior is to subclass **GenInteractionClass** and intercept MSG_GEN_GUP_INTERACTION_COMMAND. If the **InteractionCommand** sent is IC_APPLY, the application’s supplemental apply work may then be done. Then call the superclass to finish with the default apply behavior.

Note that this method cannot be used to override the default GA_SIGNAL_INTERACTION_COMPLETE behavior as that is done separately with MSG_GEN_GUP_INTERACTION_COMMAND and IC_INTERACTION_COMPLETE.

For either default or custom properties dialogs, if the properties dialog contains a large number of attributes, it may be useful to provide a means to reset the properties to their former state. This is often done with a hint such as HINT_INTERACTION_COMPLEX_PROPERTIES.

Just as you may use ATTR_GEN_TRIGGER_INTERACTION_COMMAND with IC_APPLY to provide a custom apply trigger, you may use ATTR_GEN_TRIGGER_INTERACTION_COMMAND with IC_RESET to provide a custom reset trigger. This is useful if the application needs to do some additional work when the user resets the attributes or if the application wishes to override the default reset moniker (“Reset” in OSF/Motif). You may still perform the default reset behavior (sending MSG_GEN_RESET to the UI objects) by sending MSG_GEN_GUP_INTERACTION_COMMAND with IC_RESET to the properties Interaction.

7.3.2.3 Command Interactions

GIT_COMMAND Interactions allow the user to set up parameters for and issue commands. For example, a “rename file” command in a file manager application needs to request the new name for the file from the user. This can be done with a command dialog that shows the file’s current name and has a text entry field for the file’s new name. A reply bar with a “Rename” action trigger could be provided to initiate the rename action when the user has finished entering the new name.

Command Interactions can appear as menus, dialogs, or sub-groups within a larger window. If the Interaction is displayed as a dialog, the specific UI may provide a response trigger with the IC_DISMISS **InteractionCommand** that closes the dialog without executing any commands. In OSF/Motif, this trigger is labeled “Close” or “Cancel.” You should supply your own additional command triggers yourself. You may also add custom response triggers with ATTR_GEN_TRIGGER_INTERACTION_COMMAND set to a custom **InteractionCommand**. (See “GenTrigger,” Chapter 5.)

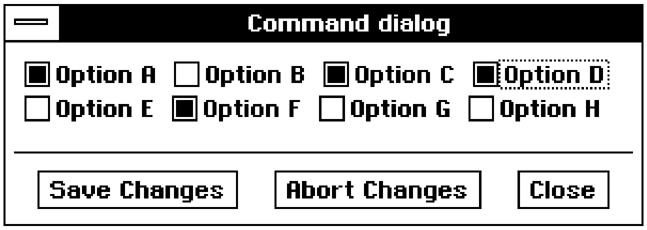


Figure 7-13 *A Command Dialog Box.*
The Specific UI supplies the “Close” trigger.

Command dialogs provide applications with a place to group related commands. They also provide a convenient place to locate UI gadgetry needed to set up parameters used by commands. A command dialog usually contains one or more command triggers in the dialog reply bar. The Specific UI will usually create a trigger to close the dialog box (with IC_DISMISS) but will not supply any command triggers; you must set up any command triggers or custom response triggers yourself.

Code Display 7-11 Using a Command Dialog Box

```
/* The Command Dialog created below contains one command trigger (RenameTrigger)
 * which sends MSG_MY_PROCESS_RENAME to the process object. */

@object GenInteractionClass RenameBox = {
    GI_visMoniker = “Rename”;
    GI_comp = RenameSource, RenameDest, RenameTrigger;
    GII_type = GIT_COMMAND;
    GII_visibility = GIV_DIALOG;
}

@object GenTextClass RenameSource = {
    /* Single-line text entry object */
}

@object GenTextClass RenameDest = {
    /* Single-line text entry object */
}

@object GenTriggerClass RenameTrigger = {
    GI_visMoniker = “Rename”;
    GTI_destination = process;
    GTI_actionMsg = MSG_MY_PROCESS_RENAME;
    GI_attrs = @default | GA_SIGNAL_INTERACTION_COMPLETE;
    HINT_SEEK_REPLY_BAR;
}
```

The Rename trigger in the above example is marked HINT_SEEK_REPLY_BAR to place it in the reply bar of the Interaction, though this is not necessary. The GA_SIGNAL_INTERACTION_COMPLETE attribute on the command trigger indicates that when the user activates this trigger, their use of the dialog is complete and the Specific UI may dismiss the dialog, if it desires. By default, non-modal command dialogs in OSF/Motif are not dismissed. You may use hints to override this behavior.

You may wish to modify this closing behavior, however. For example, if the user chooses an invalid destination name, you might not want to close that dialog; instead, the dialog should remain up and the user should be alerted that he has chosen an incorrect destination. In this case, you should remove the GA_SIGNAL_INTERACTION_COMPLETE attribute from the trigger. You should then handle the closing behavior, after validating the destination, in the MSG_MY_PROCESS_RENAME handler. To dismiss the command dialog, send MSG_GEN_GUP_INTERACTION_COMMAND with IC_DISMISS to it. (You may also use IC_INTERACTION_COMPLETE if you wish to defer to the Specific UI whether to close the dialog box.)

The Specific UI will also create a standard response trigger to dismiss the command dialog box. If some action needs to take place when the user attempts to close this dialog box, you should replace the default Close trigger by adding your own trigger to replace the IC_DISMISS **InteractionCommand**. As with IC_APPLY and IC_RESET, use ATTR_GEN_TRIGGER_INTERACTION_COMMAND with IC_DISMISS to replace the default trigger. You should also mark this trigger HINT_SEEK_REPLY_BAR if you wish it to appear in the Interaction's reply bar.

Code Display 7-12 Replacing the Default Close Trigger

```
@object GenInteractionClass RenameBox = {
    GI_visMoniker = “Rename”;
    GI_comp = RenameSource, RenameDest, RenameTrigger, RenameClose;
    GII_type = GIT_COMMAND;
    GII_visibility = GIV_DIALOG;
}

@object GenTextClass RenameSource = {
    /* Single-line text entry object */
}

@object GenTextClass RenameDest = {
    /* Single-line text entry object */
}

@object GenTriggerClass RenameTrigger = {
    GI_visMoniker = “Rename”;
    GTI_destination = process;
    GTI_actionMsg = MSG_MY_PROCESS_RENAME;
    GI_attrs = @default | GA_SIGNAL_INTERACTION_COMPLETE;
    HINT_SEEK_REPLY_BAR;
}
```

```
@object GenTriggerClass RenameClose = {
    GTI_destination = process;
    GTI_actionMsg = MSG_MY_PROCESS_CLOSE_RENAME;
    ATTR_GEN_TRIGGER_INTERACTION_COMMAND = { IC_DISMISS };
    HINT_SEEK_REPLY_BAR;
}
```

7.3.2.4 Progress Interactions

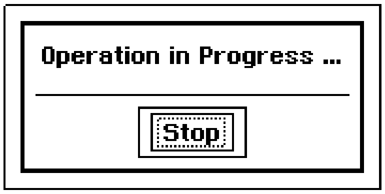
GIT_PROGRESS Interactions report the progress of an operation. You will need progress dialogs most often to show the status of an ongoing operation. Progress Interactions may appear as either a dialog or as part of a larger window. (Typically progress reports should not appear within a popup, as popups will not remain on-screen.)

If a progress Interaction takes the form of a dialog box, the Specific UI may create a response trigger with the **InteractionCommand** IC_STOP. This trigger will halt the operation (and therefore the progress report) and may close the dialog box if the Specific UI or the application desires this. In OSF/Motif this trigger is labeled “Stop.” The Specific UI will not supply a default way to close the dialog other than this IC_STOP trigger. In OSF/Motif this means that a system menu for a non-modal progress dialog will have its “Close” trigger disabled.

For example, you can use a progress dialog to report the progress of a disk format. Gadgets within the progress dialog might reflect the current state of the disk format (for example, a bar indicating the time-percentage of the operation completed). A “Stop” trigger in the reply bar allows the user to abort the disk format. Your application should intercept the **InteractionCommand** IC_STOP in this case so that you can perform any actions required in the halting operation.

A progress dialog may be either modal or non-modal, depending on the application and the operation in progress. The **GenInteractionType** GIT_PROGRESS is designed for cases where the application supports stopping an in-progress operation.

While the Specific UI will supply an IC_STOP trigger in the reply bar for halting the on-going operation, the trigger by itself does nothing. You must


Operation in Progress ...

Stop

Figure 7-14 *A Progress Dialog Box.*
The Specific UI supplies the "Stop" trigger.

either replace this IC_STOP trigger with one that performs the required halting operation, or you must provide a subclass of GenInteraction to intercept MSG_GEN_GUP_INTERACTION_COMMAND with IC_STOP. Providing a replacement IC_STOP trigger is the easiest method.

Code Display 7-13 Replacing the Default IC_STOP Trigger

```
/* This Progress Dialog shows the progress of a data transfer. Its first child
 * shows the number of packets sent in the operation, and the second child is an
 * IC_STOP trigger to halt the transfer operation. */

@object GenInteractionClass FileTransferProgress = {
    GI_visMoniker = “File Transfer Status”;
    GI_comp = FileTransferPktCount, FileTransferStop;
    GII_visibility = GIV_DIALOG;
    GII_type = GIT_PROGRESS;
}

/* The GenText object shows the title “Packets Sent” and updates the GTI_text field
 * with the current number of packets sent. */

@object GenTextClass FileTransferPktCount = {
    GI_visMoniker = “Packets sent:”;
    GTXI_text = “0”;
}
```

```
/* When the user activates the “Abort File Transfer” trigger, the trigger sends out
 * MSG_MY_PROCESS_ABORT_FILE_TRANSFER to the process. This action should also
 * close the dialog box. Since this trigger may initiate a destructive action,
 * HINT_TRIGGER_DESTRUCTIVE_ACTION makes sure that the focus does not lie on the
 * dialog box (and therefore the trigger cannot be activated by a default
 * activation method such as hitting RETURN). */

@object GenTriggerClass FileTransferStop = {
    GI_visMoniker = “Abort File Transfer”;
        /* OSF/Motif will dismiss this dialog box if marked
         * GA_SIGNAL_INTERACTION_COMPLETE because it is GIT_PROGRESS. */
    GI_attrs = @default | GA_SIGNAL_INTERACTION_COMPLETE;
    GTI_destination = process;
        /* The handler for MSG_MY_PROCESS_ABORT_FILE_TRANSFER must stop the
         * operation. You may also dismiss the dialog at that time. */
    GTI_actionMsg = MSG_MY_PROCESS_ABORT_FILE_TRANSFER;
    ATTR_GEN_TRIGGER_INTERACTION_COMMAND = { IC_STOP };
    HINT_SEEK_REPLY_BAR;
    HINT_TRIGGER_DESTRUCTIVE_ACTION;     /* Don’t place focus here. */
}
```

7.3.2.5 Notification Interactions

GIT_NOTIFICATION Interactions allow your application to send notices to the user. You should use a notification Interaction when you wish to notify the user of an event without needing to prompt the user for further information.

GIT_NOTIFICATION interactions normally appear as dialogs. If so, the Specific UI will create a response trigger with the **InteractionCommand** IC_OK. This command will close the dialog box in most Specific UIs. In OSF/Motif, this trigger is labeled “OK.” The Specific UI will not supply a way to close the dialog without responding with this trigger. In OSF/Motif, this means that the system menu in a non-modal notification dialog will have its “Close” trigger disabled.

A notification Interaction may in theory be of any visibility, but it usually takes the form of a modal dialog box. Therefore, the user should only need to acknowledge the dialog box (usually by activating the “OK” trigger).

You will usually require a notification dialog when you need to let the user know that something has occurred that may or may not require his

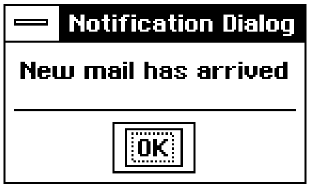


Figure 7-15 *A Notification Dialog Box.*
The Specific UI supplies the "OK" trigger.

immediate attention. For example, a notification dialog would be appropriate to notify the user that new e-mail has arrived; a notification dialog would also be appropriate to notify the user that an error has occurred.

Notifications that require immediate attention should be marked GIA_MODAL or HINT_INTERACTION_MODAL, depending on the application's dependence on their modality. Interactions that do not require immediate attention may be marked HINT_INTERACTION_NO_DISTURB so that when brought up, they do not disturb the focus or target of the application. This is especially useful for cases such as e-mail notification, where switching the focus away from another application at each new notice might become irritating to the user.

Code Display 7-14 An E-mail Notification Dialog

```
/* The Notification dialog is marked HINT_INTERACTION_NO_DISTURB so that it
 * will not grab the focus and target. The notification itself has only one child:
 * the text to display to the user. The Specific UI will also create an IC_OK
 * trigger that is used to acknowledge and dismiss the notification. */

@object GenInteractionClass NewMailNotice = {
    GI_visMoniker = "Mailbox";
    GI_comp = NewMailText;
    GII_type = GIT_NOTIFICATION;
    GII_visibility = GIV_DIALOG;
    HINT_INTERACTION_NO_DISTURB;
}
```

```
@object GenGlyphClass NewMailText = {
    GI_visMoniker = “New mail has arrived”;
}
```

By default the IC_OK trigger will just dismiss the dialog. If some other action might need to be taken when the user acknowledges the notification, you should provide a custom IC_OK trigger to replace the specific UI supplied one.

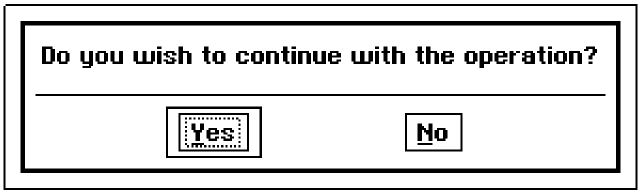
Code Display 7-15 Replacing the IC_OK Trigger

```
/* This Notification dialog now contains two explicit children: the text to
 * display to the user and a custom trigger to acknowledge the notification. */

@object GenInteractionClass NewMailNotice = {
    GI_visMoniker = “Mailbox”;
    GI_comp = NewMailText, NewMailAcknowledge;
    GII_type = GIT_NOTIFICATION;
    GII_visibility = GIV_DIALOG;
    HINT_INTERACTION_NO_DISTURB;
}

@object GenGlyphClass NewMailText = {
    GI_visMoniker = “New mail has arrived”;
}

/* When the user clicks on the Acknowledge Trigger, MSG_MY_PROCESS_NEW_MAIL_ACK is
 * sent to the process class. Marking the trigger as IC_OK replaces the default
 * trigger. Note that the Specific UI is still allowed to pick the visual moniker
 * for this trigger as its GI_visMoniker field is left blank. */

@object GenTriggerClass NewMailAcknowledge = {
    GTI_destination = process;
    GTI_actionMsg = MSG_MY_PROCESS_NEW_MAIL_ACK;
    GI_attrs = @default | GA_SIGNAL_INTERACTION_COMPLETE;
    ATTR_GEN_TRIGGER_INTERACTION_COMMAND = { IC_OK };
    HINT_SEEK_REPLY_BAR;
}
```

7.3.2.6 Affirmation Interactions

GIT_AFFIRMATION Interactions allow your application to provide the user with a choice of yes or no. GIT_AFFIRMATIONs normally appear as dialog boxes. If so, the Specific UI will create response triggers with the commands IC_YES and IC_NO. In OSF/Motif, the triggers are labeled "Yes" and "No." The Specific UI will not provide a default manner to close the dialog without responding to one of these. In OSF/Motif, this means that non-modal dialog boxes will have their "Close" triggers disabled.

For example, your application may need to prompt the user with a yes or no question (such as "Do you want to continue?"). Depending on the value returned in the **InteractionCommand**, the dialog box may continue or abort any impending operations. In most cases, you will want to use an affirmation interaction when you wish to force the user to make a choice.



Do you wish to continue with the operation?

Figure 7-16 *An Affirmation Dialog Box.*
The Specific UI supplies the "Yes" and "No" triggers

An affirmation dialog allows the user to make a single yes or no choice, without an option to ignore the choice and close the dialog. In general, this type of dialog is normally modal or used with **UserDoDialog()** to block the calling thread until the user responds. When the user chooses, the caller will be unblocked and **UserDoDialog()** will return IC_YES or IC_NO. Alternatively, you can supply replacement IC_YES or IC_NO (or both) triggers which send out custom messages when activated.

Code Display 7-16 An Affirmation Dialog for Deleting a File

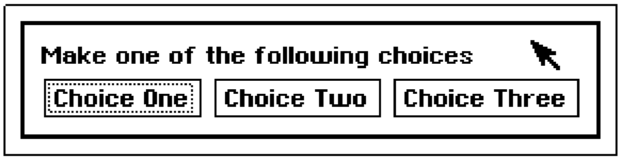
```
/* Rather than using UserDoDialog(), this dialog is simply marked GIA_MODAL.
 * This approach has the advantage of allowing any views that are run by the
 * process thread to be updated if they are exposed by the application. */
```

```
/* This dialog keeps the default IC_NO trigger to dismiss the dialog but
 * replaces the default IC_YES trigger. */

@object GenInteractionClass DeleteAffirmation = {
    GI_comp = DeleteAffirmationText, DeleteAffirmationYes;
    GII_type = GIT_AFFIRMATION;
    GII_visibility = GIV_DIALOG;
    GII_attrs = @default | GIA_MODAL;
}

@object GenGlyphClass DeleteAffirmationText = {
    GI_visMoniker = "Are you sure you want to delete this file?";
}

/* This trigger replaces the default IC_YES trigger. */

@object GenTriggerClass DeleteAffirmationYes = {
    GI_attrs = @default | GA_SIGNAL_INTERACTION_COMPLETE;
    GTI_destination = process;
    GTI_actionMsg = MSG_MY_PROCESS_DELETE_FILE;
    ATTR_GEN_TRIGGER_INTERACTION_COMMAND = {IC_YES};
    HINT_SEEK_REPLY_BAR;
}
```

7.3.2.7 Multiple Response Interactions

GIT_MULTIPLE_RESPONSE Interactions allow the UI to provide the user with a number of choices, one of which must be selected. GIT_MULTIPLE_RESPONSE Interactions normally appear as dialog boxes. The Specific UI itself supplies no standard response triggers; you must add your own response triggers. (See "Standard Response Triggers" on page 455.) There is no default provision to just close or cancel the dialog, ignoring the choice. In OSF/Motif, this means the system menu for a non-modal dialog will have its "Close" trigger disabled.

The GIT_MULTIPLE_RESPONSE type allows you to build a custom dialog box. By default, a multiple-response dialog provides no standard response triggers and no way to close the dialog. You must therefore provide any response triggers you might need (including an IC_DISMISS **InteractionCommand**). If not, the dialog will only be dismissable by the application, not by the user. In general, you should use a



Make one of the following choices

Choice One

Choice Two

Choice Three

Figure 7-17 *A Multiple Response Dialog Box.*

You must supply the triggers yourself. Note that the specific UI makes this dialog box modal, therefore preventing it from being dismissed without responding to one of the choices.

GIT_MULTIPLE_RESPONSE when you wish to prompt the user with several choices, one of which must be chosen.

Code Display 7-17 A Fragmentation Warning Dialog Box

```
@object GenInteractionClass DatabaseFragmentWarning = {
    GI_comp = DBFragmentText, DBFragmentCompact, DBFragmentContinue;
    GII_type = GIT_MULTIPLE_RESPONSE;
    GII_visibility = GIV_DIALOG;
    GII_attrs = @default | GIA_MODAL;
}

@object GenGlyphClass DBFragmentText = {
    GI_visMoniker = “Database is getting fragmented.”
}

/* As none of the following triggers are standard trigger replacements, there is no
 * need to use ATTR_GEN_TRIGGER_INTERACTION_COMMAND. */

@object GenTriggerClass DBFragmentCompact = {
    GI_visMoniker = “Run Compaction utility.”;
    GTI_destination = process;
    GTI_actionMsg = MSG_MY_PROCESS_DB_COMPACT;
    GI_attrs = @default | GA_SIGNAL_INTERACTION_COMPLETE;
    HINT_SEEK_REPLY_BAR;
}
```

```
@object GenTriggerClass DBFragmentContinue = {
    GI_visMoniker = “Continue without Compacting.”;
    GTI_destination = process;
    GTI_actionMsg = MSG_MY_PROCESS_DB_CONTINUE;
    GI_attrs = @default | GA_SIGNAL_INTERACTION_COMPLETE;
    HINT_SEEK_REPLY_BAR;
}
```

7.3.2.8 Toolboxes

Toolboxes are useful to display small icons for commonly used operations. Toolboxes are normally provided by controllers. (See “Generic UI Controllers,” Chapter 12.) In many cases toolboxes are non-modal GIT_ORGANIZATIONAL dialog boxes. Any GenInteraction may be marked HINT_TOOLBOX, however. (In some cases, you may wish a toolbox to be GIT_MULTIPLE_RESPONSE.) GeoDraw provides a toolbox to alter the properties of graphics objects. This toolbox appears at appropriate times in the GeoDraw application and will remain on-screen until the user explicitly closes it by selecting “Close” in its system menu.

Toolbox dialogs should never steal the focus or target of the application; this prevents the user from constantly having to reselect current objects. The hint HINT_TOOLBOX provides this functionality. This hint prevents a toolbox Interaction from grabbing the focus whenever an object within the toolbox is activated.

7.4 Supplemental Usage

The previous section described most of what you need to get an Interaction into your application. Much of the behavior of an Interaction is taken care of by the Specific UI. If you wish to modify this behavior, the following section details several ways in which you can alter these defaults.

7.4.1 Initiating Interactions

MSG_GEN_INTERACTION_INITIATE, MSG_GEN_INTERACTION_INITIATE_NO_DISTURB

For the most part, the Specific UI decides in what way Interactions should be initiated. For example, a popup built as a menu will create a menu title which, when activated, brings down the menu. A dialog box will create an activation trigger within the UI that, when activated, brings the dialog box on-screen. (Notice that it is meaningless to “initiate” a sub-group Interaction as these objects are not independently-displayable.)

You may wish to modify this behavior, either to prevent the user from directly initiating an Interaction, or to add additional ways to bring these Interactions on-screen.

Although in most cases the user is responsible for activating menus (by activating the menu title), an application may manually open a menu with MSG_GEN_INTERACTION_INITIATE.

A GIV_DIALOG Interaction may be initiated in several ways:

- In the default case, the specific UI will create an activation trigger whose sole function is to bring up the dialog box. The trigger will appear as a child of the dialog’s parent, in the spot within the UI that the dialog would normally reside were it not independently displayable.
- A dialog box may be set GIA_NOT_USER_INITIATABLE to prevent the building of the default trigger. Any Interaction set non-user initiatable should be sent a MSG_GEN_INTERACTION_INITIATE or a MSG_GEN_INTERACTION_INITIATE_NO_DISTURB to bring the dialog on-screen. You may also put Interactions on the GAGCNLT_WINDOWS GCN list to automatically bring them on-screen when an application is launched. Any Interaction must be fully usable and enabled to be brought up in such a manner. Note: If a dialog box is also marked GIA_INITIATED_VIA_USER_DO_DIALOG, it should be initiated with **UserDoDialog()**, *not* MSG_GEN_INTERACTION_INITIATE. These methods are exclusive.
- A dialog box may be brought up with **UserDoDialog()** if it is set GIA_INITIATED_VIA_USER_DO_DIALOG. This routine is useful when you wish to block the calling thread until the user responds to the dialog box. A dialog box initiated via **UserDoDialog()** should also be marked

GIA_NOT_USER_INITIATABLE. Note that this dialog box may only be brought up with **UserDoDialog()**; it cannot be brought up with MSG_GEN_INTERACTION_INITIATE.

Within most specific UIs a trigger will appear whose sole function is to initiate the dialog. Activating the trigger will bring the dialog box on-screen. You can prevent the display of this default trigger by setting the Interaction GIA_NOT_USER_INITIATABLE in its *GII_attrs* instance field. (See “GenInteraction Attributes” on page 402).

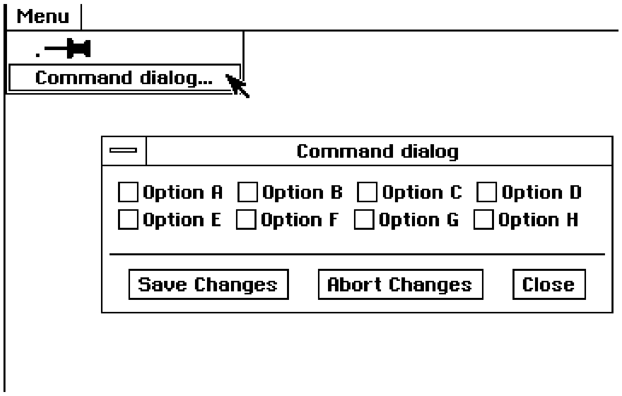


Figure 7-18 *Dialog with Activation Trigger*

The dialog box (Command dialog), when placed within a menu, creates a menu item trigger that initiates the dialog box.

Your GenInteraction subclass may intercept MSG_GEN_INTERACTION_INITIATE to determine when a dialog is being brought on-screen and then perform any needed actions at that time. This is useful if you need to update the UI objects within the dialog before building its visual implementation. You should always call the superclass *after* setting any of this initial data to avoid unnecessary visual updates of the dialog box. MSG_GEN_INTERACTION_INITIATE_NO_DISTURB may also be subclassed, but there is little need to intercept it.

■ MSG_GEN_INTERACTION_INITIATE

```
void      MSG_GEN_INTERACTION_INITIATE();
```

This message brings an Interaction on-screen, giving it focus and target if possible. The Interaction must be both within the generic tree and set GS_USABLE before receiving this message. The Interaction must also be fully usable and enabled before it can be initiated. Interactions marked GIA_INITIATED_VIA_USER_DO_DIALOG should not be sent this message. To initiate those dialogs, you must use **UserDoDialog()**. (See “Thread Blocking Routines” on page 442.)

You may intercept this message to determine when a dialog is coming on-screen. This is useful if you need to set up UI gadgetry in the dialog to show some initial status. The gadgetry should be set up before calling the superclass to avoid unnecessary visual updating of the object.

Source: Unrestricted.

Destination: Any GS_USABLE GenInteraction, though usually only meaningful for GIV_DIALOG interactions.

Parameters: None.

Return: Nothing.

Interception: May be intercepted to find out when an Interaction is appearing on-screen. This is useful for cases in which the UI gadgetry of a dialog box needs to be set up before being visually built. The superclass should be called after setting up the UI to avoid unnecessary visual changes once the interaction is on-screen.

■ MSG_GEN_INTERACTION_INITIATE_NO_DISTURB

```
void        MSG_GEN_INTERACTION_INITIATE_NO_DISTURB();
```

This message brings an Interaction on-screen but does not give the dialog the focus or target. This message may place the dialog behind other windows. The Interaction must be both within the generic tree and set GS_USABLE before receiving this message. The Interaction must also be fully usable and enabled before it can be initiated. Interactions marked GIA_INITIATED_VIA_USER_DO_DIALOG should not be sent this message. To initiate those dialogs, you must use **UserDoDialog()**. (See “Thread Blocking Routines” on page 442.)

Source: Unrestricted.

Destination: Any GS_USABLE GenInteraction, though usually only meaningful for GIV_DIALOG Interactions. Not allowed for GIA_MODAL or GIA_INITIATED_VIA_USER_DO_DIALOG Interactions.

Parameters: None.

Return: Nothing.

Interception: May be intercepted to find out when an Interaction is appearing on-screen. This is useful for cases in which the UI gadgetry of a dialog box needs to be set up before being visually built. The superclass should be called after setting up the UI to avoid unnecessary visual changes once the interaction is on-screen.

7.4.2 Dismissing Interactions

You may dismiss dialogs and popups under application control by sending MSG_GEN_GUP_INTERACTION_COMMAND with the **InteractionCommand** IC_DISMISS to them. This will force dialogs to close even if the user has such a dialog pinned (in Specific UIs that support pinning). Dialogs may also be dismissed when the user activates reply bar response triggers with the GA_SIGNAL_INTERACTION_COMPLETE attribute set. (See “GenClass,” Chapter 2.)

You may intercept MSG_GEN_GUP_INTERACTION_COMMAND to determine when a dialog is about to be dismissed. The handler for this message should check the incoming **InteractionCommand** to make sure it is IC_DISMISS before doing whatever work is needed. You should make sure to pass any non-IC_DISMISS **InteractionCommand** immediately to the superclass. The handler may or may not need to call the superclass for IC_DISMISS, depending on what behavior the handler is adding.

7.4.3 Modality for Dialogs

MSG_GEN_INTERACTION_TEST_INPUT_RESTRICTABILITY

By default, GIV_DIALOG GenInteractions appear as non-modal windows. You may specify a dialog Interaction to appear as modal by setting either the GIA_MODAL attribute in the *GII_attrs* instance data field or by adding HINT_INTERACTION_MODAL in the object’s vardata. When a modal dialog is displayed, input to all other parts of the application will be ignored until the dialog is dismissed. In OSF/Motif, modal dialogs will appear with thick borders.

You should use GIA_MODAL dialogs when the application depends on an Interaction being modal to operate correctly. For example, if a dialog shows

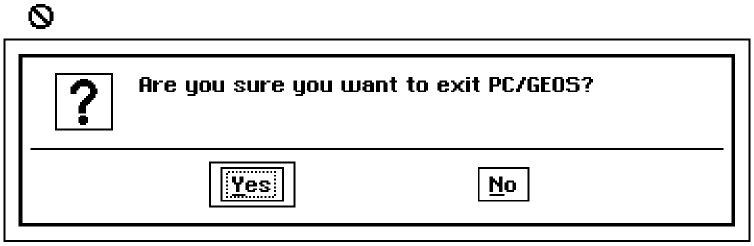

Are you sure you want to exit PC/GEOS?

Figure 7-19 *A modal dialog box*
Whenever the cursor roams outside the bounds of the modal dialog box, input to other parts of the application is blocked.

information about the current selection but the application cannot update that information if the user were to change the current selection, you should use a GIA_MODAL dialog.

You should use HINT_INTERACTION_MODAL dialogs for cases where the modality of a dialog simplifies the interface between the application and the user but is not needed for the application's correct operation. The Interaction's modality in this case will make the UI clearer and simpler for the user to understand. For example, a dialog for setting obscure options that is brought up from a command dialog might be marked HINT_INTERACTION_MODAL so that the user is forced to set the options before returning to the command dialog. This will prevent the user from using this dialog box in a context where the options are not self-evident. As with all hints, the Specific UI may decide to ignore the hint and implement a HINT_INTERACTION_MODAL dialog with a non-modal dialog.

You may also initiate a modal dialog with **UserDoDialog()**. Such a dialog must have both the GIA_INITIATED_VIA_USER_DO_DIALOG and GIA_MODAL attributes set. Using this routine not only blocks input to other parts of the application but blocks the calling thread's execution until the dialog is dismissed. **UserDoDialog()** is useful for displaying notifications where the user's response is required before the application's thread may continue. For more information, see "Thread Blocking Routines" on page 442.

Dialogs displayed with **UserStandardDialog()** and **UserStandardDialogOptr()** are also modal. Both will block the calling thread until the dialog is dismissed. (See “Thread Blocking Routines” on page 442.)

Some modal dialogs will have subcomponents such as pop-up lists that must be interactable while the dialog is modal. The GenApplication object running the modal dialog will query the dialog to test both its input restrictions and its interactability in these cases; it uses two messages, described below.

■ MSG_GEN_INTERACTION_TEST_INPUT_RESTRICTABILITY

```
Boolean  MSG_GEN_INTERACTION_TEST_INPUT_RESTRICTABILITY(
         optr   obj);
```

This message is called on a modal window by a GenApplication object to find out whether or not the various input restricting mechanisms (input hold-up, input ignore, busy states, etc.) should be overridden or not.

Source: Part of the input flow/modality mechanism—sent by the GenApplication object to the modal window.

Destination: The modal GenInteraction object.

Parameters: obj The optr of the windowed object that may or may not be restricted.

Return: Will return *true* if input restrictions should be overridden, false otherwise. A *true* return value indicates that mouse and keyboard input will flow to the object regardless of hold-up, ignore, or busy states.

Interception: May be intercepted to allow the passed windowed object to override its input restrictions in certain cases.

7.4.4 Managing Input

ATTR_GEN_INTERACTION_OVERRIDE_INPUT_RESTRICTIONS,
ATTR_GEN_INTERACTION_ABIDE_BY_INPUT_RESTRICTIONS,
HINT_INTERACTION_DEFAULT_ACTION_IS_NAVIGATE_TO_NEXT_FIELD

Typically, specific UIs will determine how input (primarily keyboard input) affects GenInteractions and their components. For example, many UIs will have the Tab key navigate from the active component of a dialog box to the next available component.

HINT_INTERACTION_DEFAULT_ACTION_IS_NAVIGATE_TO_NEXT_FIELD allows you to trade the normal “default action” (the action typically causing the dialog to send its “apply” message) for navigation. For example, if hitting the return key normally invoked an “Apply” trigger, this hint would instead cause it to navigate to the next field in the dialog box.

Two other attributes allow a GenInteraction to either disobey or abide by certain default input restrictions (modality, input hold-up, input ignore, and busy states). ATTR_GEN_INTERACTION_OVERRIDE_INPUT_RESTRICTIONS allows the GenInteraction to override the input restrictions; ATTR_GEN_INTERACTION_ABIDE_BY_INPUT_RESTRICTIONS allows the GenInteraction to override default behavior ignoring those restrictions.

7.4.5 Thread Blocking Routines

Occasionally, you may need a response from the user before continuing with a thread of execution. In these cases, you should use a dialog box to prompt the user for the needed response. However, you also need a means to block the thread of execution until the user responds to this dialog box. The routines **UserDoDialog()**, **UserStandardDialog()**, and **UserStandardDialogOptr()** provide this functionality.

7.4.5.1 UserDoDialog()

`UserDoDialog(), UserCreateDialog(), UserDestroyDialog()`

You may bring dialog boxes on-screen with several routines. The most common and easiest to use of these routines is **UserDoDialog()**. **UserDoDialog()** only operates on GIV_DIALOG Interactions set both GIA_MODAL and GIA_INITIATED_WITH_USER_DO_DIALOG.

This routine, when passed the optr of a dialog box, will bring the dialog Interaction on-screen. In addition to bringing up a dialog, however, **UserDoDialog()** will also block the calling thread until a response trigger in the dialog is activated by the user. When this happens, **UserDoDialog()** returns a value representing the response trigger selected. This return value is usually an **InteractionCommand**.

The dialog may contain any UI gadgetry but must have response triggers with valid **InteractionCommand**s to terminate the dialog box. These triggers should have a null output and message and have ATTR_GEN_TRIGGER_INTERACTION_COMMAND vardata set to the proper **InteractionCommand**s. (See “GenTrigger,” Chapter 5.) This **InteractionCommand** will be the response value returned by **UserDoDialog()** when that trigger is activated. This can be one of the predefined **InteractionCommand**s or an application-defined one with IC_CUSTOM_START.

The response triggers should also have the attribute GA_SIGNAL_INTERACTION_COMPLETE set to ensure that the dialog will be dismissed when they are activated. (This may be omitted to gain manual control over dismissal over the dialog, but you still must provide a way to dismiss the dialog.) Any response triggers in the dialog should have the hint HINT_SEEK_REPLY_BAR to place the triggers within the dialog’s reply bar, but this is not necessary.

Because the calling thread is blocked by **UserDoDialog()**, views run by that thread will not be updated when exposed.

Code Display 7-18 Using UserDoDialog()

```
/* This dialog box asks for confirmation before beginning a delete file operation.
 * Therefore, it is advisable to block threads before beginning this operation. The
 * dialog using UserDoDialog() should be marked both GIA_MODAL and
 * GIA_INITIATED_VIA_USER_DO_DIALOG. */

@object GenInteractionClass ConfirmDeleteBox = {
    GI_comp = ConfirmDeleteText;
    GII_type = GIT_AFFIRMATION;
    GII_visibility = GIV_DIALOG;
    GII_attrs = @default | GIA_INITIATED_VIA_USER_DO_DIALOG | GIA_MODAL;
}

@object GenTextClass ConfirmDeleteText = {
    GTI_text = “Are you sure you want to delete this file?”;
}

/* This dialog box is displayed through a normal routine call with
 * the optr of the dialog box as its one argument. */
```

```
/* Check for positive response. */
if (UserDoDialog(ConfirmDeleteBox) == IC_YES) {
    /* delete file. */
}
```

UserDoDialog() may also return IC_NULL to indicate that the modal dialog has been dismissed by the system, so it is always a good idea to check for and act on positive responses even if only one response is possible (such as in a GIT_NOTIFICATION dialog box).

UserCreateDialog() duplicates a dialog box to initiate later with **UserDoDialog()**. Typically, the dialog box duplicated is within a template object block; the dialog box must be both not GS_USABLE and not attached to the generic tree when created. The dialog box must also be marked GIA_INITIATED_VIA_USER_DO_DIALOG.

The template block that contains the dialog box and its children must be sharable and read-only. **UserCreateDialog()** duplicates a template dialog box, attaches the dialog box to the GenApplication object and sets it fully GS_USABLE; it may then be called with **UserDoDialog()**. When you no longer have a need for the dialog box, send it **UserDestroyDialog()**.

These routines are useful for conserving memory space; they only take up space when actually being used. In some cases, you may need to use these routines. For example, within libraries, dialog boxes must be duplicated before being used because multiple applications may require their own copy of the dialog box template.

Code Display 7-19 Using UserCreateDialog(), UserDestroyDialog()

```
/*
 * The template dialog box must not be GS_USABLE. The object must also be marked
 * GIA_INITIATED_VIA_USER_DO_DIALOG. The block must be sharable, read-only, and
 * the top GenInteraction must not be linked into the generic tree.
 */

@object GenInteractionClass MyDialogTemplate = {
    GI_visMoniker = “Template”;
    GI_states = @default & ~GS_USABLE;
    GII_visibility = GIV_DIALOG;
    GII_attrs = @default | GIA_INITIATED_VIA_USER_DO_DIALOG |
        GIA_NOT_USER_INITIATABLE | GIA_MODAL;
    GII_type = GIT_NOTIFICATION;
    GI_comp = @NotificationGlyph;
}

@method SomeProcessClass, MSG_BRING_UP_DUPLICATED_DIALOG
{
    optr        newDialog;

    newDialog = UserCreateDialog(@MyDialogTemplate);

    if (UserDoDialog(@newDialog) == IC_OK) {
        /*** code ***/
    }

    UserDestroyDialog(@newDialog);
}
```

7.4.5.2 UserStandardDialog()

`UserStandardDialog(), UserStandardDialogOptr(), CustomDialogBoxFlags`

UserStandardDialog() displays standardized dialog boxes. The dialog is standardized in that it has a text area, an icon glyph representing the type of situation that caused the dialog to be displayed, and one or more response triggers. Like **UserDoDialog()**, **UserStandardDialog()** blocks the calling thread until the user activates one of the response triggers. Unlike **UserDoDialog()**, however, **UserStandardDialog()** does not need an application-defined dialog box. **UserStandardDialog()** builds a dialog box at run-time following the specifications passed.

UserStandardDialog() passes a number of parameters:

◆ **CustomDialogBoxFlags**
This bitfield stores the **CustomDialogType** (CDBF_DIALOG_TYPE) and the **GenInteractionType** (CDBF_INTERACTION_TYPE) to display the Interaction with, along with several miscellaneous flags

CDBF_SYSTEM_MODAL indicates that the dialog box brought up by **UserDoDialog()** should not only be application modal but also system modal.

CDBF_DESTRUCTIVE_ACTION indicates that an affirmative response by the user to the dialog box denotes a destructive action, and thus should not be made the default.

CDBF_DIALOG_TYPE indicates the type of situation creating the dialog box. This type determines the icon glyph that represents what caused the dialog to be displayed. The available types (**CustomDialogType**) are:

CDT_QUESTION: Ask the user a question;
CDT_WARNING: Warn the user of a potential problem;
CDT_NOTIFICATION: Notify the user of some event;
CDT_ERROR: Report an error to the user.

The Specific UI will determine the appropriate icon glyph to use for each dialog type. For example, in OSF/Motif, a CDT_QUESTION icon glyph is a graphic question mark.

The system will also issue a beep when a CDT_ERROR dialog is displayed.

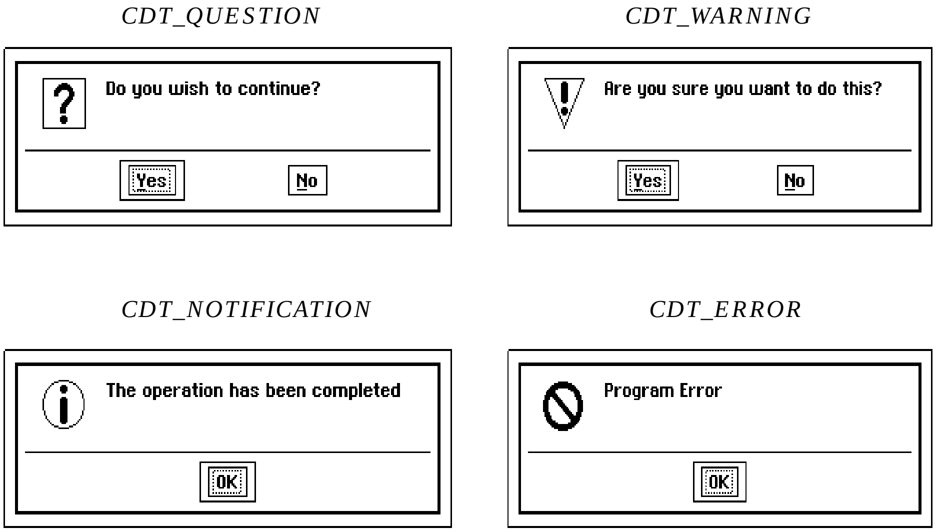
CDBF_INTERACTION_TYPE indicates the type of GenInteraction being initiated. This type specifies what response triggers should be built and is a sub-set of the supplied **GenInteractionType** enums. The available types are:

GIT_NOTIFICATION
The specific UI will supply a standard response trigger that has the IC_OK response value.

GIT_AFFIRMATION
The specific UI will supply standard response triggers that have the IC_YES and IC_NO response values.

GIT_MULTIPLE_RESPONSE
The application must provide its own trigger monikers and response values. If this value is set, you will have to pass the array of custom triggers.

- Dialog string (and *arg1* and *arg2*)
  This is the string to display in the dialog box text area. The string can have up to two parameters. These parameters are other strings that are passed to **UserStandardDialog()** in *arg1* and *arg2*. They will replace all occurrences of *arg1* and *arg2* in the dialog string. This is useful for including filenames or other variable text in the dialog string.



CDT_QUESTION

Do you wish to continue?

CDT_WARNING

Are you sure you want to do this?

CDT_NOTIFICATION

The operation has been completed

CDT_ERROR

Program Error

Figure 7-20 *Standard Dialog Boxes*

Each CustomDialogType provides its own standard graphic glyph and standard response triggers. The CDT_ERROR dialog also provides an audible warning.

- Response trigger monikers and values
  If GIT_MULTIPLE_RESPONSE is used, this indicates how many triggers and what labels and response values the response triggers should have.
- Help context
  The help context to use for this dialog box layer.

Code Display 7-20 Using UserStandardDialog()

```
/*
 * This simple example uses no help context, custom triggers, or string arguments.
 */

if ((UserStandardDialog( (char *)0,
                         (char *)0,
                         (char *)0,
                         (char *)0,
                         “Do you wish to continue?”,
                         ((CDT_QUESTION << CDBF_DIALOG_TYPE_OFFSET) |
                        (GIT_AFFIRMATION << CDBF_INTERACTION_TYPE_OFFSET))
                        ) == IC_YES)) {
        /* code to perform on a positive response. */
    }

else {
        /* code to perform on a negative response. */
    }
```

You may also use **UserStandardDialogOptr()** for cases in which the strings are referenced through optrs rather than pointers.

Code Display 7-21 A ConfirmDeleteBox with explicit monikers

```
/* For this case, since we want to provide explicit monikers, we must use the
 * GIT_MULTIPLE_RESPONSE interaction type. Using this allows us to pass in the
 * monikers and response values for the response triggers for the dialog. This is
 * done by passing a pointer to a table consisting of the number of triggers in the
 * dialog and a StandardDialogResponseTriggerEntry for each trigger. Each entry
 * contains an optr of the moniker to use and the response value for the trigger.
 * The moniker may be simple text or a graphics string. The response value may be
 * one of the predefined InteractionCommands or an application-defined value based
 * on IC_CUSTOM_START. */

@visMoniker ConfirmYesMoniker = “Delete this file”;

@visMoniker ConfirmNoMoniker = “Skip this file”;

/* Create a table to hold the trigger data. */

static const StandardDialog2ResponseTriggerTable confirmResponseTable [] = {
    2,                          /* SD2RTT_numTriggers */
        /* WRT_trigger1 */
    {ConfirmYesMoniker,         /* SDRTE_moniker */
        IC_YES},                /* SDRTE_responseValue */
        /* WRT_trigger2 */
    {ConfirmNoMoniker,          /* SDRTE_moniker */
        IC_NO}                  /* SDRTE_responseValue */
};

/* Display the dialog with UserStandardDialog(). */
```

```
if (UserStandardDialog(  (char *)0,
                         (char *)&confirmResponseTable,
                         (char *)0,
                         (char *)0,
                         “Are you sure you want to delete this file?”,
                         ((CDT_QUESTION << CDBF_DIALOG_TYPE_OFFSET) |
                         /* interaction type - application supplied trigger */
                         (GIT_MULTIPLE_RESPONSE << CBDF_INTERACTION_TYPE_OFFSET)))
                == IC_YES) {
        /* delete file */
}
```

Code Display 7-22 A IC_CUSTOM_START Interaction

```
#define SAVE_CHANGES             IC_CUSTOM_START+0
#define ABORT_CHANGES            IC_CUSTOM_START+1
#define CANCEL_CLOSE             IC_CUSTOM_START+2

@visMoniker CloseSaveMoniker = “Save Changes”;
@visMoniker CloseAbortMoniker = “Abort Changes”:
@visMoniker CloseCancelMoniker = “Cancel Close”;

static const StandardDialog3ResponseTriggerTable closeResponseTable [] = {
     3,                        /* SD2RTT_numTriggers */
        /* WRT_trigger1 */
    {CloseSaveMoniker,                 /* SDRTE_moniker */
    IC_YES},                           /* SDRTE_responseValue */
        /* WRT_trigger2 */
    {CloseAbortMoniker,                /* SDRTE_moniker */
    IC_YES},                           /* SDRTE_responseValue */
        /* WRT_trigger3 */
    {CloseCancelMoniker,               /* SDRTE_moniker */
    IC_NO}                             /* SDRTE_responseValue */
};

closeWithChangesResponse =
        (UserStandardDialog(
                (char *)0,
                (char *)&closeResponseTable),
                (char *)0,
                (char *)0,
                ((CDT_QUESTION << CDBF_DIALOG_TYPE_OFFSET) |
                /* interaction type - application supplied triggers */
                (GIT_MULTIPLE_RESPONSE << CDBF_INTERACTION_TYPE_OFFSET)));
```

```
switch (closeWithChangesResponse) {
    case SAVE_CHANGES:
        /* save changes */
    case ABORT_CHANGES:
        /* abort changes */
    case IC_CANCEL_CLOSE:
        /* cancel close */
    case IC_NULL:
        /* IC_NULL is always a potential response */
}
```

7.5 Interaction Commands

Much of the flexibility of GenInteraction lies in its ability to recognize and process a variety of special **InteractionCommand**s. These commands are data types that the Specific UI and the Interaction can use to decide what actions to take. Depending on the **InteractionCommand**, the UI may perform several pre-defined actions. Much of this functionality is automatic through the inclusion of standard response triggers within a dialog box.

Some **GenInteractionType**s provide one or more triggers for your use when the Interaction is built as a dialog box. These default triggers are automatically provided without your needing to create and attach triggers to the Interaction itself. Triggers provided in this manner are known as standard response triggers. Standard response triggers usually appear within an area of the dialog box known as a reply bar.

Each standard response trigger has a specific **InteractionCommand** associated with it. These triggers perform their actions by sending out their **InteractionCommand** with the message MSG_GEN_GUP_INTERACTION_COMMAND to themselves when activated. The message will travel up the tree to the first GIV_DIALOG Interaction, where it will be handled. See “InteractionCommand Types” on page 451 for the pre-defined **InteractionCommand**s.

7.5.1 InteractionCommand Types

Many of these **InteractionCommand**s are sent out by standard response triggers provided with the Specific UI. Different types of Interactions may provide different standard response triggers with **InteractionCommand**s. For example, the "OK" trigger for a GIT_NOTIFICATION dialog box in OSF/Motif sends out the **InteractionCommand** IC_OK.

The **InteractionCommand** represents the actions that can be performed on an Interaction. The predefined commands are as follows:

- IC_NULL
  This **InteractionCommand** is a special case for alerting a **UserDoDialog()** or **UserStandardDialog()** dialog box that it is being dismissed because of system shutdown. Because of thread blocking, you should always check for this case to occur in any **UserDoDialog()** routines. You should never set any custom triggers to this **InteractionCommand**.

- IC_DISMISS
  This command instructs the UI to dismiss the Interaction. An Interaction receiving this command will come down whether or not the user has selected the Interaction to remain up (such as by pinning the dialog box).

- IC_APPLY
  This command instructs the UI to apply properties selected within a GIT_PROPERTIES Interaction. This **InteractionCommand** causes MSG_GEN_APPLY to be sent to the Interaction's children.

- IC_RESET
  This command instructs the UI to reset properties within a GIT_PROPERTIES Interaction. This **InteractionCommand** causes MSG_GEN_RESET to be sent to the Interaction's children.

- IC_OK
  This command instructs the UI to signal an acknowledgment to a GIT_NOTIFICATION Interaction.

- IC_YES
  This command instructs the UI to signal a positive response to a GIT_AFFIRMATION Interaction.

- IC_NO
  This command instructs the UI to signal a negative response to a GIT_AFFIRMATION Interaction.

- IC_STOP
  This command instructs the UI to halt a GIT_PROGRESS Interaction.

- IC_EXIT
  This command instructs the UI to exit the application (not just the window). This **InteractionCommand** is only supported for GenTriggers under GIV_POPUP GenInteractions. This value should only be used with ATTR_GEN_TRIGGER_INTERACTION_COMMAND.

- IC_HELP
  This command instructs the UI to bring up the proper help context for this dialog. This value should only be used with ATTR_GEN_TRIGGER_INTERACTION_COMMAND.

- IC_INTERACTION_COMPLETE
  This command notifies the UI that an interaction has been completed. This **InteractionCommand** is sent out by any trigger with the attribute GA_SIGNAL_INTERACTION_COMPLETE *before* the trigger's main action has been sent out. Note that this **InteractionCommand** is *always* sent out in addition to another IC. You should never set any custom triggers to this **InteractionCommand**.

IC_DISMISS, IC_APPLY, IC_RESET, IC_OK, IC_YES, IC_NO, and IC_STOP correspond to standard response triggers provided with the Specific UI for dialogs of various **GenInteractionType**. (See "GenInteraction Types" on page 399.)

You may replace any of these standard response triggers by adding your own triggers with an **InteractionCommand**. You should do this by setting the trigger's ATTR_GEN_TRIGGER_INTERACTION_COMMAND vardata to the proper **InteractionCommand** to replace. (See "Replacing Default Triggers" on page 457.)

7.5.1.1 Adding Custom Response Triggers

In some cases, you may want to add your own response triggers. In many of these cases, a trigger tied to one of the predefined **InteractionCommand**s will be sufficient for your needs. If a dialog box does not contain one of the pre-defined response triggers by default, you can add any of these yourself.

You are not limited to one of these predefined types, however. **UserDoDialog()** and GIT_MULTIPLE_RESPONSE **UserStandardDialog()** allow you to define your own **InteractionCommand** types. Applications can define their own **InteractionCommand** by starting an enumeration at the special value IC_CUSTOM_START. This is to avoid conflicting with other predefined values, which still have their usual meaning to the dialog.

Code Display 7-23 Using IC_CUSTOM_START

```
/* You may define values directly. */

#define IC_SAVE_FILE IC_CUSTOM_START+1;
#define IC_KILL_FILE IC_CUSTOM_START+2;

/* You may also use typedef to enumerate those values. */

typedef enum {
    IC_SAVE_FILE = IC_CUSTOM_START+1,
    IC_KILL_FILE = IC_CUSTOM_START+2
} MyInteractionCommand;

/* Any triggers set up with these InteractionCommands would appear like so: */

@object GenTriggerClass MyTrigger = {
    GI_visMoniker = “Save File”;
    ATTR_GEN_TRIGGER_INTERACTION_COMMAND = { IC_SAVE_FILE };
    HINT_SEEK_REPLY_BAR;
}

/* An Interaction invoked with UserDoDialog() (with MyTrigger as one of its
 * children) will return IC_SAVE_FILE when that trigger is activated. Your
 * application can then perform any required actions. */
```

7.5.2 Dialog Control

MSG_GEN_GUP_INTERACTION_COMMAND,
MSG_GEN_INTERACTION_ACTIVATE_COMMAND

There are two ways of delivering an **InteractionCommand** to an Interaction. The first is through the **GenClass** message MSG_GEN_GUP_INTERACTION_COMMAND. In most cases, this is the message you will use to deliver commands to your dialog boxes. For example, you may send MSG_GEN_GUP_INTERACTION_COMMAND with IC_DISMISS to dismiss a dialog box. The second involves sending MSG_GEN_INTERACTION_ACTIVATE_COMMAND with one of the specified **InteractionCommand** types.

The system sends this message whenever the user activates a response trigger with null action fields. The system will pass the command in ATTR_GEN_TRIGGER_INTERACTION_COMMAND vardata with the message up the generic tree until handled by the first GIV_DIALOG Interaction. Handling may include unblocking and returning a **UserDoDialog()** response value, applying or resetting properties, and dismissing, depending on the **InteractionCommand**. Note that if an action is specified in the trigger's instance data, MSG_GEN_GUP_INTERACTION_COMMAND will not be sent—the specified action will instead.

In OSF/Motif, dialogs perform the following default activity with their various **InteractionCommand**s:

IC_DISMISS This command dismisses the dialog, resets gadgets if GIT_PROPERTIES and modal (i.e. “Cancel” trigger).

IC_APPLY This command applies gadgets (whether or not GIT_PROPERTIES), then unblocks and returns a **UserDoDialog()** value if needed.

IC_RESET This command resets gadgets (whether or not GIT_PROPERTIES), then unblocks and returns a **UserDoDialog()** value if needed.

IC_INTERACTION_COMPLETE
This command dismisses, depending on dialog context, the dialog box and unblocks and returns a **UserDoDialog()** value if needed.

All other **InteractionCommands** will just unblock and return a **UserDoDialog()** value if needed.

GIV_DIALOG GenInteractions are the most likely recipients of any **InteractionCommand**s. Popup Interactions only support IC_DISMISS and IC_INTERACTION_COMPLETE and will pass other **InteractionCommand**s up the generic tree. Any other Interactions (e.g. sub-groups) will always pass MSG_GEN_GUP_INTERACTION_COMMAND up the generic tree.

◆Objects

MSG_GEN_INTERACTION_ACTIVATE_COMMAND is a higher-level function. When sent to a dialog GenInteraction with a relevant **InteractionCommand**, it will find the standard trigger for that **InteractionCommand** and activate it. This can be a Specific UI-supplied trigger or a replacement trigger supplied by the application. If no such trigger exists, the message sends MSG_GEN_GUP_INTERACTION_COMMAND to perform the default handling. This message is only supported by dialog Interactions.

■ MSG_GEN_INTERACTION_ACTIVATE_COMMAND

```
void      MSG_GEN_INTERACTION_ACTIVATE_COMMAND(
          word   command);
```

This message activates a GenTrigger having the passed command in its ATTR_GEN_TRIGGER_INTERACTION_COMMAND; if no such trigger exists, this message calls MSG_GEN_GUP_INTERACTION_COMMAND to the Interaction itself with the supplied command value. Note that **InteractionCommand** type IC_INTERACTION_COMPLETE should not be used as this is a notification rather than a command.

Source: Unrestricted.

Destination: Any non GIT_ORGANIZATIONAL GenInteraction object.

Parameters: *command* InteractionCommand to activate (except IC_INTERACTION_COMPLETE and IC_NULL).

Return: Nothing.

Interception: May be intercepted to change or supplement default functionality of standard or custom interaction commands. Standard response triggers will use this message, so intercepting this message will allow changing the default behavior of these triggers without having to replace them with your own triggers.

7.5.3 Standard Response Triggers

The Specific UI provides standard response triggers for certain **GenInteractionType**, when built as dialog boxes. These standard response triggers contain an appropriate **InteractionCommand** and usually a visual moniker chosen by the Specific UI.

The standard response triggers (and their visual monikers under OSF/Motif) for each *GII_type* are shown in Table 7-1.

Table 7-1 *Standard Response Triggers*

Interaction Type	Trigger Type	Moniker
GIT_NOTIFICATION	IC_OK	"OK"
GIT_AFFIRMATION	IC_YES	"Yes"
	IC_NO	"No"
GIT_PROGRESS	IC_STOP	"Stop"
GIT_COMMAND	IC_DISMISS	"Close" or "Cancel"
GIT_PROPERTIES	IC_APPLY	"Apply" or "OK"
	IC_RESET	"Reset"
	IC_DISMISS	"Close" or "Cancel"

The Specific UI automatically places these standard response triggers in a reply bar within the dialog. In most cases, you will not have to deal directly with any of these standard response triggers; the Specific UI will create and manage them. In other cases, you must either supply additional response triggers or replace the existing default response triggers. Additional triggers must be supplied for GIT_COMMAND and GIT_MULTIPLE_RESPONSE Interactions. You may replace existing triggers to customize their behavior or even just to change their visual monikers.

You should add HINT_SEEK_REPLY_BAR to any additional triggers you supply so that they will be placed in the reply bar area of the Interaction (though this is not necessary). If you add more than one trigger, their order within the Interaction's children will be the order they appear in the reply bar. However, supplemental reply bar triggers will appear to the right of any IC_OK, IC_YES, IC_STOP, or IC_APPLY and to the left of any IC_DISMISS, IC_RESET, or IC_NO trigger.

Certain triggers may act as default triggers in the reply bar. Default triggers do not need to be explicitly activated (i.e. clicked on) to send out their action. Default triggers may also be activated if the user performs a default activation (normally hitting "Return" or double-clicking on an object within the dialog). Normally this default activation trigger will be the first trigger in the reply bar. Default triggers may be highlighted in some manner by the

Specific UI. You can override this behavior by marking any reply trigger of your choosing with the hint HINT_DEFAULT_DEFAULT_ACTION.

The Specific UI gives HINT_DEFAULT_DEFAULT_ACTION to the IC_APPLY, IC_OK and IC_YES triggers in any reply bars it creates. (These are normally the first triggers anyway.) For example, the "OK" trigger in a GIT_NOTIFICATION dialog may be activated by either clicking on the trigger or hitting RETURN.

Whenever a dialog is initially brought on-screen, the trigger marked with HINT_DEFAULT_DEFAULT_ACTION will act as the default activation trigger. However, activating any other trigger will make that trigger the default activation trigger the next time the dialog is brought on-screen. You can avoid this by adding HINT_PREVENT_DEFAULT_OVERRIDES on the GenInteraction itself. This will allow the HINT_DEFAULT_DEFAULT_ACTION trigger to always remain the default trigger.

7.5.4 Replacing Default Triggers

You may replace a trigger supplied by the Specific UI by adding your own custom GenTrigger (as a child of the GenInteraction) with its ATTR_GEN_TRIGGER_INTERACTION_COMMAND set to the proper **InteractionCommand** of the trigger to replace. You should also mark such triggers with HINT_SEEK_REPLY_BAR to place them within the reply bar, though this is not required. If there is no such hint, the trigger will appear within the body of the dialog.

You should set the trigger to perform whatever action your application requires (including the function of the trigger being replaced, if applicable). If necessary, the action handler should also perform the default handling of the replaced trigger. A moniker need not be supplied as the Specific UI will choose one based on the **InteractionCommand** and the context of the dialog. You may supply your own moniker.

In fact, in some cases you might want to merely change the default moniker of a standard response trigger without changing the functionality of the trigger itself. In this case, you should provide an override trigger with ATTR_GEN_TRIGGER_INTERACTION_COMMAND. You should then add your new visual moniker to this replacement but leave the action fields blank. The specific UI will assign any trigger set up with ATTR_GEN_TRIGGER_INTERACTION_COMMAND and a blank action to have the default behavior of the replaced trigger.

Code Display 7-24 Replacing a Standard Response Trigger

```
/* The GenInteractionType GIT_NOTIFICATION, when built as a dialog, automatically
 * contains a standard response trigger to send out the InteractionCommand IC_OK.
 * You can replace this trigger with one of your own by adding a trigger with that
 * InteractionCommand. Your trigger will override the system default. */

@object GenInteractionClass MyInteraction = {
    GII_type = GIT_NOTIFICATION;
    GII_visibility = GIV_DIALOG;
    GII_attrs = @default | GIA_NOT_USER_INITIATABLE | GIA_MODAL;
    GI_comp = MyGlyph, MyResponseTrigger;
}

@object GenGlyphClass MyGlyph = {
    GI_visMoniker = “New Mail has arrived”;
}

/* MyResponseTrigger replaces the IC_OK default trigger for the Notification dialog
 * box. In this case, we will only change the visual moniker of the trigger. By
 * leaving the output fields blank, we cause the system to send
 * MSG_GEN_GUP_INTERACTION_COMMAND with IC_OK. (This is the default behavior.)
 * HINT_SEEK_REPLY_BAR is used to place this trigger within the reply bar of the
 * notification dialog. */

@object GenTriggerClass MyResponseTrigger = {
    GI_visMoniker = “OK, I acknowledge”;
    ATTR_GEN_TRIGGER_INTERACTION_COMMAND = { IC_OK };
    HINT_SEEK_REPLY_BAR;
}
```

ATTR_GEN_TRIGGER_INTERACTION_COMMAND also performs a special function in dialogs displayed with **UserDoDialog()**. Whenever the user completes an Interaction in a **UserDoDialog()** dialog box by activating one of the response triggers, the **InteractionCommand** stored in the ATTR_GEN_TRIGGER_INTERACTION_COMMAND of the activated trigger will be returned. You can use this return value to determine which trigger the user activated. (See “Thread Blocking Routines” on page 442.)

7.5.5 Triggers Completing Interactions

The **GenClass** attribute GA_SIGNAL_INTERACTION_COMPLETE is used to indicate that a response trigger completes a single user interaction within the dialog that contains it. Depending on the context (modality, type, hints) of the dialog, the Specific UI may or may not dismiss the dialog box.

For example, the "Yes" trigger (IC_YES) in a GIT_AFFIRMATION dialog might dismiss a dialog if the trigger is also marked GA_SIGNAL_INTERACTION_COMPLETE because the user has given the dialog final information. An "Apply" trigger (IC_APPLY) marked GA_SIGNAL_INTERACTION_COMPLETE might not dismiss its dialog because it might be useful to use the dialog to repeatedly apply different properties. This behavior depends on the Specific UI.

All standard response triggers provided by the Specific UI with the various **GenInteractionType**, except for the IC_RESET trigger), have GA_SIGNAL_INTERACTION_COMPLETE set. The IC_RESET trigger normally does not indicate the end of user interaction with the dialog and thus should not dismiss the dialog under any circumstances.

In OSF/Motif, a trigger marked GA_SIGNAL_INTERACTION_COMPLETE will dismiss the dialog containing the trigger if the dialog is not of **GenInteractionType** GIT_PROPERTIES or GIT_COMMAND, or if the dialog is modal. A non-modal GIT_PROPERTIES or GIT_COMMAND dialog is not dismissed when its response triggers are activated. If you do wish to dismiss the dialog in such cases, you can mark the dialog box with HINT_INTERACTION_SINGLE_USAGE. This hint tells the Specific UI that it expects the user will only use the dialog box once each time it is brought up. You may also want to do this if the dialog box is infrequently used.

Similarly, HINT_INTERACTION_FREQUENT_USAGE allows a dialog to stay up even if one of its GA_SIGNAL_INTERACTION_COMPLETE triggers would normally bring it down. In most cases, however, it is advisable to remove the GA_SIGNAL_INTERACTION_COMPLETE attribute on whichever triggers should not dismiss the Interaction. This is preferred over adding HINT_INTERACTION_FREQUENT_USAGE because the Specific UI may ignore the hint, but it cannot ignore the state of its GA_SIGNAL_INTERACTION_COMPLETE bit.

When using HINT_INTERACTION_FREQUENT_USAGE, you should ensure that the dialog is dismissed. In most cases, this can be done by sending MSG_GEN_GUP_INTERACTION_COMMAND with IC_DISMISS to the Interaction. This can be done in an action handler for a response trigger.

GenValue

8

GenValueClass allows the user to retrieve or set a numerical value for use in your application. The GenValue object offers a display for the current value and the means to either increase or decrease this value. The GenValue object also manages the range that this value may fall between and the action to send out when a value has been selected.

Before reading this chapter, you should be familiar with **GenClass** and with message passing. (See "GenClass," Chapter 2.)

8.1 GenValue Features

Depending upon the specific UI, the GenValue object can look like one of many things. It may be a spin gadget, a slider, a dial, or some other gadget. A GenValue typically allows the user to set a value within a specified range through keyboard or mouse input. For example, many GenValue objects in OSF/Motif are comprised of a text field showing the current value, an incrementor and a decrementor, and possibly a moniker. (Scroll bars are also GenValues, though they have very different appearance.)

In OSF/Motif, the user can enter the value by keyboard input into the text field or by clicking the mouse on the incrementor or decrementor.

The GenValue provides the following capabilities:

- A user interface object to enter values into your application.
- Application defined minimum and maximum values.
- The increment value to increase or decrease values by.
- Ability for the value to be represented in a variety of forms, including integer, decimal, or a unit of distance.
- An action to apply any value to.

The user may change a value in a GenValue in one of two ways. It can be entered with the keyboard by typing directly into the display area, or it can be entered graphically by using the arrows, dials, or sliders that the specific

UI provides. To enter a value by typing directly into the text field, first the user clicks on the area, bringing up a cursor. Then a value may be typed into the GenValue. To enter a value graphically, the user clicks on either the incrementor or decrementor, thus changing the value of the GenValue.

8.1

Once the value is changed, the value may become set immediately (if the object is operating in *immediate* mode), and the GenValue will perform its action (if any) every time the value is changed. As an alternative, the GenValue may be operating in *delayed* mode, and the value will not be set until a MSG_GEN_APPLY is received through an "Apply" or "OK" button. The mode (delayed or immediate) of the GenValue is controlled by the parent. Usually a GenValue will operate in immediate mode unless it is placed as a child of a GIT_PROPERTIES GenInteraction.

GenValue only allows the user to set a value within a specific range; this range consists of a minimum and maximum value that the GenValue may display. These values are set by the application and cannot be directly modified by the user, but they can be changed by your application. If the user enters a value above the maximum, the GenValue simply takes on its maximum possible value. If the user enters a value below the minimum, the GenValue takes on its least possible value.

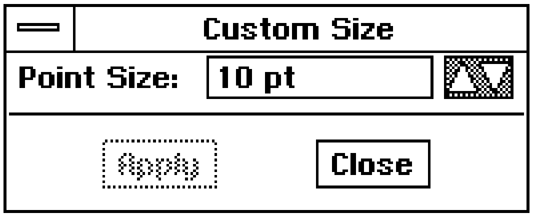


Figure 8-1 *A GenValue*

This image is of a GenValue in OSF/Motif. It is in the point display mode and is used to customize the font sizes in GeoWrite.

In addition to the limits upon the range of values entered by the user, the GenValue also has a defined increment value. The increment value is the specific amount to increment or decrement the GenValue's value when using graphical input. The user cannot directly modify this value, but it can be changed by your application.

The GenValue object always displays a numerical value within its text field. The units of that numerical value may be set by your application. By default, any value within a GenValue is an integer, but that value could also be a decimal value, or even a unit of distance such as centimeters, points, or inches. The GenValue automatically provides whatever notation is required by the display units (decimal points, the words “pt” and “cm,” or any other distance notation in the text field). (See Figure 8-1.)

8.2 GenValue Instance Data

The GenValue contains instance fields that affect the current value, the display units of that value, and the action to take when any value operation has been completed. These instance fields are listed in Code Display 8-1. Remember, in addition to these instance data variables, there are also the instance fields inherited from the GenValue’s superclass, **GenClass**.

Code Display 8-1 GenValue Instance Data

```
/* The instance data fields for GenValue are shown below. Those that are records
 * of flags have their default flags shown with other possible flags listed below.
 * Other fields are shown with their default values. */

    @instance WWFixedAsDWord     GVLI_value = MakeWWFixed(0.0);
    @instance WWFixedAsDWord     GVLI_minimum = MakwWWFixed(0.0);
    @instance WWFixedAsDWord     GVLI_maximum = MakeWWFixed(32766);
    @instance WWFixedAsDword     GVLI_increment = MakeWWFixed(1.0);
    @instance GenValueStateFlags GVLI_stateFlags = 0;

/* GenValueStateFlags */
    typedef ByteFlags GenValueStateFlags;
    #define GVSF_INDETERMINATE   0x80
    #define GVSF_MODIFIED        0x40
    #define GVSF_OUT_OF_DATE     0x20

    @instance GenValueDisplayFormatGVLI_displayFormat = GVDF_INTEGER;

/* GenValueDisplayFormat */
    typedef ByteEnum GenValueDisplayFormat;
    #define GVDF_INTEGER 0
    #define GVDF_DECIMAL 1
    #define GVDF_POINTS 2
```

```
#define GVDF_INCHES 3
#define GVDF_CENTIMETERS 4
#define GVDF_MILLIMETERS 5
#define GVDF_PICAS 6
#define GVDF_EUR_POINTS 7
#define GVDF_CICEROS 8
#define GVDF_POINTS_OR_MILLIMETERS 9
#define GVDF_POINTS_OR_CENTIMETERS 10
```

```
@instance optr              GVLI_destination;
@instance Message           GVLI_applyMsg = 0;
```

GVLI_value is the current numerical value of the GenValue. By default, it is an integer constant defined by the application. Depending on the *GVLI_displayFormat*, it can be interpreted as an integer, a decimal, or distance unit.

GVLI_maximum is the maximum possible value that the GenValue may display. This value may be positive or negative. The default value is 32766.

GVLI_minimum is the minimum possible value that the GenValue may display. This value may be positive or negative. The default value is zero.

GVLI_increment is the value to increment (or decrement) the GenValue when its value is changed by UI controls rather than direct text input. The default value is one.

GVLI_stateFlags specifies the **GenValueStateFlags** for the GenValue to operate under. These flags affect whether the data within the GenValue is indeterminate (not necessarily true) or modified (changed since the last MSG_GEN_APPLY).

GVLI_displayFormat specifies the units of measurement (**GenValueDisplayFormat**) that the numerical value represents. By default, this is an integer, but it could also be a decimal or a unit of measurement (such as inches, points, or centimeters).

GVLI_applyMsg is the message to send out upon GenValue changes (i.e. whenever it receives MSG_GEN_APPLY). There is no default message.

GVLI_destination is the object or process to send the message upon GenValue changes. This can be an optr to an object or a valid **TravelOption**. See "System Classes," Chapter 1. There is no default destination object.

Code Display 8-2 GenValue Optional Attribute Fields

```
@vardata Message     ATTR_GEN_VALUE_STATUS_MSG;
@vardata word        ATTR_GEN_VALUE_DECIMAL_PLACES;
@vardata WWFixed     ATTR_GEN_VALUE_METRIC_INCREMENT;
@vardata optr        ATTR_GEN_VALUE_RUNS_ITEM_GROUP;
    @reloc ATTR_GEN_VALUE_RUNS_ITEM_GROUP, 0 optr;
@vardata void        ATTR_GEN_VALUE_SET_MODIFIED_ON_REDUNDANT_SELECTION;
```

ATTR_GEN_VALUE_STATUS_MSG sets a status message for a GenValue. A status message allows your object to communicate with other objects when changes occur without sending out its apply message (*GVLI_applyMsg*).

ATTR_GEN_VALUE_DECIMAL_PLACES specifies the number of decimal places to display to the right of the decimal point if the *GVLI_displayFormat* allows fractional quantities.

ATTR_GEN_VALUE_METRIC_INCREMENT allows you to specify a particular metric increment to use besides the default if the *GVLI_displayFormat* is GVDF_POINTS_OR_MILLIMETERS or GVDF_INCHES_OR_CENTIMETERS.

ATTR_GEN_VALUE_RUNS_ITEM_GROUP links a GenValue to a GenItemGroup.

ATTR_GEN_VALUE_SET_MODIFIED_ON_REDUNDANT_SELECTION specifies that the GenValue should be marked modified whether or not a change in the value has occurred. This will result in that value being applied whenever it receives a MSG_GEN_APPLY. (The default behavior for when no change in state occurs, no message will be sent out.)

Code Display 8-3 GenValue Hints

```
@vardata void        HINT_VALUE_INCREMENTABLE;
@vardata void        HINT_VALUE_NOT_INCREMENTABLE;

@vardata void        HINT_VALUE_NAVIGATE_TO_NEXT_FIELD_ON_RETURN_PRESS;
@vardata Message     HINT_VALUE_CUSTOM_RETURN_PRESS;
```

```
    @vardata WWFixedAsDWord     HINT_VALUE_DISPLAYS_RANGE;
    @vardata GenValueIntervals  HINT_VALUE_DISPLAY_INTERVALS;
    @vardata void       HINT_VALUE_CONSTRAIN_TO_INTERVALS;

typedef struct {
        word    GVI_numMajorIntervals;
        word    GVI_numMinorIntervals;
} GenValueIntervals;

    @vardata void       HINT_VALUE_SHOW_MIN_AND_MAX;
    @vardata void       HINT_VALUE_IMMEDIATE_DRAG_NOTIFICATION;
    @vardata void       HINT_VALUE_DELAYED_DRAG_NOTIFICATION;
    @vardata void       HINT_VALUE_ORIENT_HORIZONTALLY;
    @vardata void       HINT_VALUE_ORIENT_VERTICALLY;

    @vardata void       HINT_VALUE_ANALOG_DISPLAY;
    @vardata void       HINT_VALUE_DIGITAL_DISPLAY;
    @vardata void       HINT_VALUE_NO_DIGITAL_DISPLAY;
    @vardata void       HINT_VALUE_NO_ANALOG_DISPLAY;
    @vardata void       HINT_VALUE_NOT_DIGITALLY_EDITABLE;
    @vardata void       HINT_VALUE_DO_NOT_MAKE_LARGER_ON_PEN_SYSTEMS;
```

HINT_VALUE_INCREMENTABLE and HINT_VALUE_NOT_INCREMENTABLE specify whether increment (and decrement) gadgets are appropriate for this GenValue. By default, GenValues are incrementable.

HINT_VALUE_CUSTOM_RETURN_PRESS stores a Message to send out if the user hits return within the textual portion of a GenValue. Normally, this action triggers the interaction default (sending the apply message).

HINT_VALUE_DISPLAYS_RANGE indicates that this GenValue displays a range of values between its minimum and maximum. In most cases, this suggests that the GenValue use sliders or some other UI gadget that is able to show the width of a range. It is up to the specific UI to support range values. If this hint is present, *GVLI_value* refers to the starting point of the range of values, whose width is stored in the **WWFixedAsDWord** value here. The maximum *GVLI_value* in this case would be *GVLI_maximum* minus the range. If this hint is not present, the range "length" is presumed to be zero, even in gadgets that specify a range length by default.

HINT_VALUE_DISPLAY_INTERVALS indicates that intervals should be displayed along an object's range. This hint is used most often in analog (e.g. slider) type displays. If possible, hash marks will appear along the

GenValue's display at the intervals specified by **GenValueIntervals**. This structure stores entries for both major intervals and minor intervals. In general, minor intervals will have smaller tick marks than major intervals.

```
typedef struct {
      word GVI_numMajorIntervals;
      word GVI_numMinorIntervals;
} GenValueIntervals;
```

If either *GVI_numMajorIntervals* or *GVI_numMinorIntervals* is zero, only one set of marks will appear.

HINT_VALUE_CONSTRAIN_TO_INTERVALS suggests that the value within *GVLI_value* constrain itself to the *GVI_numMinorIntervals* interval within HINT_VALUE_DISPLAY_INTERVALS.

HINT_VALUE_SHOW_MIN_AND_MAX instructs the GenValue to display its minimum and maximum values, if possible.

HINT_VALUE_IMMEDIATE_DRAG_NOTIFICATION instructs the GenValue to send out its status and/or apply messages constantly during a drag operation (each time the value changes). It is up to the specific UI to support this behavior.

HINT_VALUE_DELAYED_DRAG_NOTIFICATION instructs the GenValue to delay sending out status and/or apply message until the user releases the mouse after the drag operation. It is up to the specific UI to support this behavior.

HINT_VALUE_ORIENT_HORIZONTALLY instructs the specific UI to arrange an analog GenValue, if available, in the horizontal dimension.

HINT_VALUE_ORIENT_VERTICALLY instructs the specific UI to arrange an analog GenValue, if available, in the vertical dimension.

HINT_VALUE_ANALOG_DISPLAY indicates that the GenValue should be displayed in an analog fashion, if applicable. HINT_VALUE_DIGITAL_DISPLAY indicates that the GenValue should be displayed in a digital fashion (i.e. numerically). Similarly, HINT_VALUE_NO_DIGITAL_DISPLAY and HINT_VALUE_NO_ANALOG_DISPLAY indicate that a digital or analog display is not appropriate.

HINT_VALUE_NOT_DIGITALLY_EDITABLE instructs the UI to disallow editing of the text within a numerical GenValue. This hint is only applicable if some other UI means of changing the value is available. To remove a numeric display of values entirely, use HINT_VALUE_NO_DIGITAL_DISPLAY.

HINT_VALUE_DO_NOT_MAKE_LARGER_ON_PEN_SYSTEMS instructs the specific UI to avoid expanding a GenValue object to accept pen input, if possible. By default, text areas of GenValues grow larger under pen systems to allow ink strokes. This hint usually indicates that non-ink means of entering values is available.

8.3 GenValue Basics

The GenValue instance fields can be set to specific values in your Goc file, and they can also be modified by your application at run-time. This section describes how to set and modify these fields in your Goc file.

8.3.1 The Value

GVLI_value, MSG_GEN_VALUE_SET_VALUE,
MSG_GEN_VALUE_SET_INTEGER_VALUE, MSG_GEN_VALUE_GET_VALUE

The *GVLI_value* instance field stores the current numerical value of the GenValue. You may set an initial value for the GenValue to appear with by setting this instance field in your Goc file. This value is a fixed point number; use **MakeWWFixed** to create this fixed point number in your instance data.

Any user changes on the value within the text field will not affect *GVLI_value* until MSG_GEN_APPLY applies that value. If the GenValue operates in delayed mode, it will be marked modified in its *GVLI_stateFlags* whenever a user change occurs; those changes will be applied when the GenValue receives a MSG_GEN_APPLY. In most cases, however, a GenValue operates in immediate mode, which will result in an immediate change in *GVLI_value*.

Code Display 8-4 Setting an Initial Value

```
/* This GenValue will appear with the initial integer value of two. MakeWWFixed
 * creates a fixed point value. */

@object GenValueClass MyValue = {
    GI_visMoniker = “My Value”;
    GVLI_value = MakeWWFixed(2.0);
}
```

GenValueClass provides several messages to change the value without user control. MSG_GEN_VALUE_SET_VALUE sets this numeric value to a passed fixed point value; this fixed point value may be any integer or decimal value. MSG_GEN_VALUE_SET_INTEGER_VALUE is a simpler message which sets *GVLI_value* to an integer value passed. Neither of these messages mark the GenValue modified; you can do this with MSG_GEN_VALUE_SET_MODIFIED_STATE.

■ MSG_GEN_VALUE_SET_VALUE

```
void    MSG_GEN_VALUE_SET_VALUE(
        WWFixedAsDWord      value,
        Boolean             indeterminate);
```

This message sets the *GVLI_value* field of the GenValue to the passed *value*. This message clears a GenValue’s modified state in its *GVLI_stateFlags*. To mark the GenValue modified send MSG_GEN_VALUE_SET_MODIFIED_STATE after sending this message.

Source: Unrestricted. This message is also used internally when responding to user actions.

Destination: Any GenValue object.

Parameters:

value	The fixed point value to set *GVLI_value* to. If you only need an integral value, consider using MSG_GEN_VALUE_SET_INTEGER_VALUE instead.
indeterminate	TRUE to mark the GenValue indeterminate, FALSE to mark it non indeterminate.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_VALUE_SET_INTEGER_VALUE

```
void      MSG_GEN_VALUE_SET_INTEGER_VALUE(
          word                value,
          Boolean             indeterminate);
```

This message sets the *GVLI_value* to the passed integer (word-sized) value. *GVLI_value* will then contain this value in its high (integral) word and zero in its low (fractional) word. The modified state of the GenValue will be cleared.

Source: Unrestricted. This message is also used internally when responding to user actions.

Destination: Any GenValue object.

Parameters: *value* The signed integer value to set *GVLI_value* to.

indeterminate TRUE to mark the GenValue indeterminate, FALSE to mark it non indeterminate.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_VALUE_GET_VALUE

```
WWFixedAsDword MSG_GEN_VALUE_GET_VALUE();
```

This message returns the value stored in the GenValue's *GVLI_value* instance field. This returned value will be a fixed point number.

Source: Unrestricted.

Destination: Any GenValue object.

Parameters: None.

Return: The fixed point numerical value of the *GVLI_value* instance field.

Interception: Generally not intercepted.

■ MSG_GEN_VALUE_GET_INTEGER_VALUE

```
@alias (MSG_GEN_VALUE_GET_VALUE)
                 word MSG_GEN_VALUE_GET_INTEGER_VALUE();
```

This message returns the integral portion of the GenValue's value.

Source: Unrestricted.

Destination: Any GenValue object.

Parameters: None.

Return: The integral value of the *GVLI_value* instance field.

Interception: Generally not intercepted.

8.3.2 The Minimum and Maximum

`GVLI_minimum, GVLI_maximum, MSG_GEN_VALUE_GET_MAXIMUM, MSG_GEN_VALUE_SET_MAXIMUM, MSG_GEN_VALUE_GET_MINIMUM, MSG_GEN_VALUE_SET_MINIMUM`

The maximum (*GVLI_maximum*) and minimum (*GVLI_minimum*) values of a GenValue constrain the value of *GVLI_value*. The maximum is the greatest value that the GenValue's *GVLI_value* can have and can be any fixed point number from -32767 up to 32767. The minimum is the least value that the GenValue's *GVLI_value* can have (including negative numbers) and can be any fixed point number from 32767 down to -32767. The maximum value must always be greater than or equal to the minimum value; otherwise, an error will occur.

The maximum and minimum values of the GenValue also can be examined and modified by the application. This is helpful if you need to use the same GenValue for two functions with different ranges of values. You can use the GenValue for one function, change the bounds, and use it for another function.

MSG_GEN_VALUE_GET_MINIMUM and MSG_GEN_VALUE_GET_MAXIMUM return the values of *GVLI_minimum* and *GVLI_maximum* respectively. This value is in fixed point format.

MSG_GEN_VALUE_SET_MINIMUM and MSG_GEN_VALUE_SET_MAXIMUM set the values of *GVLI_minimum* and *GVLI_maximum* respectively. If this new minimum or maximum places the current *GVLI_value* outside the valid value range, the value will be adjusted to fall within the current range.

■ MSG_GEN_VALUE_GET_MAXIMUM

```
WWFixedAsDWord MSG_GEN_VALUE_GET_MAXIMUM()
```

This message returns the fixed point value within the *GVLI_maximum* instance field of the GenValue.

Source: Unrestricted.

Destination: Any GenValue object.

Parameters: None.

Return: The fixed point maximum value in *GVLI_maximum*.

Interception: Generally not intercepted.

■ MSG_GEN_VALUE_SET_MAXIMUM

8.3

```
void      MSG_GEN_VALUE_SET_MAXIMUM(
          WWFixedAsDWord      value);
```

This message sets the *GVLI_maximum* field of the GenValue to the passed fixed point value. If you set a new maximum that places the current value (in *GVLI_value*) above that maximum, *GVLI_value* will be adjusted to this maximum.

Source: Unrestricted. This message is also used internally when the GenValue is being built.

Destination: Any GenValue object.

Parameters: *value* fixed point value to set *GVLI_maximum* to.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_VALUE_GET_MINIMUM

```
WWFixedAsDWord MSG_GEN_VALUE_GET_MINIMUM()
```

This message returns the fixed point value within the *GVLI_minimum* instance field of the GenValue.

Source: Unrestricted.

Destination: Any GenValue object.

Parameters: None.

Return: The fixed point minimum value in *GVLI_minimum*.

Interception: Generally not intercepted.

■ MSG_GEN_VALUE_SET_MINIMUM

```
void    MSG_GEN_VALUE_SET_MINIMUM(
        WWFixedAsDword      value);
```

This message sets the *GVLI_minimum* instance field of the GenValue to the passed fixed point value. If you set a new minimum that places the current value (in *GVLI_value*) below that minimum, *GVLI_value* will be adjusted to this minimum.

Source: Unrestricted. This message is also used internally when the GenValue is being built.

Destination: Any GenValue object.

Parameters: *value* fixed point value to set *GVLI_minimum* to.

Return: Nothing.

Interception: Generally not intercepted.

8.3.3 The Increment

`GVLI_increment, MSG_GEN_VALUE_GET_INCREMENT, MSG_GEN_VALUE_SET_INCREMENT, HINT_VALUE_INCREMENTABLE, HINT_VALUE_NOT_INCREMENTABLE`

The *GVLI_increment* instance field stores the fixed point increment value for a GenValue. The increment value is the amount by which the current value (in *GVLI_value*) may increase or decrease when that value is changed by UI controls. The increment value can be any positive fixed point value between one and 65535.

If *GVLI_increment* is greater than the possible range of the GenValue (the distance between minimum and maximum), then incrementing or decrementing the GenValue will toggle *GVLI_value* between its maximum and minimum.

Code Display 8-5 Setting Minimum, Maximum, Increment Values

```
@object GenValueClass MyValue = {
    GI_visMoniker = “My Value”;
    GVLI_value = MakeWWFixed(1.0);
    GVLI_minimum = MakeWWFixed(-100.0);
    GVLI_maximum= MakeWWFixed(100.0);
    GVLI_increment = MakeWWFixed(5.0);
}
```

You may change the fixed point value of this increment with MSG_GEN_VALUE_SET_INCREMENT. You may also return the current increment stored within *GVLI_increment* with MSG_GEN_VALUE_GET_INCREMENT. You may wish to change a GenValue's increment one GenValue is being used for multiple roles and must change its increment value.

By default, all GenValues are incrementable. If you do not wish to have your GenValue provide UI controls for incrementing (or decrementing) its value, add HINT_VALUE_NOT_INCREMENTABLE in its instance data. HINT_VALUE_INCREMENTABLE provides the default behavior. (You may still increment or decrement the value manually with MSG_GEN_VALUE_INCREMENT or MSG_GEN_VALUE_DECREMENT.)

■ MSG_GEN_VALUE_SET_INCREMENT

```
void    MSG_GEN_VALUE_SET_INCREMENT (
        WWFixedAsDWord      value);
```

This message sets the *GVLI_increment* field to the passed *value*.

Source: Unrestricted. This message is also used internally when the GenValue is being built.

Destination: Any GenValue object.

Parameters: *value* fixed point value to set *GVLI_increment* to.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_VALUE_GET_INCREMENT

```
WWFixedAsDWord MSG_GEN_VALUE_GET_INCREMENT();
```

This message returns the value of the *GVLI_increment* field of the GenValue.

Source: Unrestricted.

Destination: Any GenValue object.

Parameters: None.

Return: The fixed point value of *GVLI_increment*.

Interception: Generally not intercepted.

8.3.4 GenValue States

```
GVLI_stateFlags, MSG_GEN_VALUE_SET_INDETERMINATE_STATE,
MSG_GEN_VALUE_IS_INDETERMINATE,
MSG_GEN_VALUE_SET_MODIFIED_STATE,
MSG_GEN_VALUE_IS_MODIFIED,
ATTR_GEN_SET_MODIFIED_ON_REDUNDANT_SELECTION
```

GVLI_stateFlags stores the current state of the GenValue. There are two **GenValueStateFlags**:

- GVSF_INDETERMINATE
  This flag specifies that the value within the GenValue is indeterminate (may or may not be true). In most cases, you will not need to set this flag.

- GVSF_MODIFIED
  This flag specifies that the value within the GenValue has changed since it last received a MSG_GEN_APPLY. The handler for MSG_GEN_APPLY checks whether this flag is set before sending out the GenValue's *GVLI_applyMsg*.

- GVSF_OUT_OF_DATE
  This flag specifies that the value within the GenValue is out of date with what the user has typed in. This is distinct from the GVSF_MODIFIED state; while the user is typing an a value ("123" for example) the typed value may be temporarily out of range, or incomplete. In this case, the value would be marked GVSF_OUT_OF_DATE and GVSF_MODIFIED. Notice that if the value were incremented or decremented, it would be marked GVSF_MODIFIED, but not GVSF_OUT_OF_DATE because the

value is legal and presumable valid for operations. This flag is most useful when telling status messages whether a value should be used.

GenValues are normally marked as not modified anytime their state is set with an external message, marked modified whenever the user interacts with them, and marked not modified after receiving MSG_GEN_APPLY. MSG_GEN_VALUE_SET_MODIFIED_STATE allows you to control the modified state of a GenValue outside of these events.

8.3

You may set a GenValue's indeterminate or modified state with MSG_GEN_VALUE_SET_INDETERMINATE_STATE or MSG_GEN_VALUE_SET_MODIFIED_STATE, respectively.

To check whether a GenValue is indeterminate or modified, use MSG_GEN_VALUE_IS_INDETERMINATE or MSG_GEN_VALUE_IS_MODIFIED.

You may also mark a GenValue GVSF_OUT_OF_DATE by sending it MSG_GEN_VALUE_SET_OUT_OF_DATE.

■ MSG_GEN_VALUE_SET_INDETERMINATE_STATE

```
void      MSG_GEN_VALUE_SET_INDETERMINATE_STATE(
          Boolean             indeterminateState);
```

This message sets the indeterminate state for a GenValue. Pass TRUE to mark the GenValue indeterminate, FALSE to mark it not indeterminate. The GenValue will not be marked modified after this message.

Source: Unrestricted.

Destination: Any GenValue object.

Parameters: *indeterminateState*

TRUE to set the GVSF_INDETERMINATE flag in the GenValue's *GVLI_stateFlags*,
FALSE to clear the GVSF_INDETERMINATE flag.

Interception: Generally not intercepted.

■ MSG_GEN_VALUE_IS_INDETERMINATE

```
Boolean   MSG_GEN_VALUE_IS_INDETERMINATE();
```

This message checks whether a GenValue is indeterminate.

Source: Unrestricted.

Destination: Any GenValue object.

Return: TRUE if GenValue is indeterminate, FALSE if it is not.

Interception: Generally not intercepted.

■ MSG_GEN_VALUE_SET_MODIFIED_STATE

```
void     MSG_GEN_VALUE_SET_MODIFIED_STATE(
         Boolean              modifiedState);
```

This message sets the modified state for a GenValue. Pass TRUE to mark the GenValue modified, FALSE to mark it not modified.

Source: Unrestricted.

Destination: Any GenValue object.

Parameters: *modifiedState*

TRUE to set the GVSF_MODIFIED flag in the GenValue's *GVLI_stateFlags*,
FALSE to clear the GVSF_MODIFIED flag.

Interception: Generally not intercepted.

■ MSG_GEN_VALUE_IS_MODIFIED

```
Boolean  MSG_GEN_VALUE_IS_MODIFIED();
```

This message checks whether a GenValue has been modified since the last MSG_GEN_APPLY.

Source: Unrestricted.

Destination: Any GenValue object.

Return: TRUE if GenValue is modified, FALSE if it is not.

Interception: Generally not intercepted.

■ MSG_GEN_VALUE_SET_OUT_OF_DATE

```
void     MSG_GEN_VALUE_SET_OUT_OF_DATE();
```

This message sets a GenValue's GVSF_OUT_OF_DATE flag.

Source: Unrestricted.

Destination: Any GenValue object.

Interception: Generally not intercepted.

8.3.5 Display Formats

```
GVLI_displayFormat, MSG_GEN_VALUE_SET_DISPLAY_FORMAT,
MSG_GEN_VALUE_GET_DISPLAY_FORMAT,
ATTR_GEN_VALUE_METRIC_EQUIVALENT,
ATTR_GEN_VALUE_DECIMAL_PLACES
```

In addition to displaying an integer numerical value, a GenValue may also display numerical values of several other formats. These formats may be any one of the **GenValueDisplayFormat** enumerations provided in **GenValueClass**. The allowed enumerations of type **GenValueDisplayFormat** are:

GVDF_INTEGER
GVDF_DECIMAL
GVDF_POINTS
GVDF_INCHES
GVDF_CENTIMETERS
GVDF_MILLIMETERS
GVDF_PICAS
GVDF_EUR_POINTS
GVDF_CICEROS
GVDF_POINTS_OR_MILLIMETERS
GVDF_INCHES_OR_CENTIMETERS

GVLI_displayFormat controls how the values of the GenValue will be represented. For example, if the display format is in centimeters, the display will include the text “cm” after the numerical value; if the display includes a fractional part, a decimal point will be present. These display formats will also convert the values (which are stored as points) into the proper distance units for the textual display.

By default, the value’s fractional portion will be displayed using 3 places to the right of the decimal point. You may alter this number of places with ATTR_GEN_VALUE_DECIMAL_PLACES. You may only choose a number of decimal places between zero to four (inclusive), because of the limited text space offered within a GenValue.

Your GenApplication object contains instance data which specifies whether the application is being run under US or metric units. This setting may affect the display of your units depending on the particular **GenValueDisplayFormat**.

GVDF_INTEGER displays the value as an integer (the high word of the fixed point value) and ignores any fractional part.

GVDF_DECIMAL displays the value as a decimal value.

GVDF_POINTS displays the value in points (1/72 of an inch) regardless of whether metric or US units are specified for the application.

GVDF_INCHES displays the value in inches regardless of whether metric or US units are specified for the application.

GVDF_CENTIMETERS displays the value in centimeters regardless of whether metric or US units are specified for the application.

GVDF_MILLIMETERS displays the value in millimeters regardless of whether metric or US units are specified for the application.

GVDF_PICAS displays the value as a distance in picas. One pica is equal to 12 US Points, or 1/6 of an inch.

GVDF_EUR_POINTS displays the value as a distance in European Points. One european point is about equal to 1.0656 US Points.

GVDF_CICEROS displays the value as a distance in Ciceros. One Cicero is equal to 12 European Points.

GVDF_POINTS_OR_MILLIMETERS and GVDF_INCHES_OR_CENTIMETERS are special cases. These display formats display the value in points (or inches) if US units are specified for the application; the value will be represented in millimeters (or centimeters) if metric units are instead specified.

Importantly, *all* distance units (inches, picas, centimeters, etc.) store their values as Points (1/72 inch). The system automatically converts these values (in Points) into the proper units of the GenValue's *GVLI_displayFormat* when it displays the numerical value within the textual display.

Code Display 8-6 Setting a Distance Display Format

```
@object GenValueClass MyValue = {
    GI_visMoniker = “My Value”;
    GVLI_displayFormat = GVDF_INCHES;
/* For an initial value of 1 inch, the equivalent value in Points is 72. */
    GVLI_value = MakeWWFixed(72.0);

/* For an initial value of 1/2 inch, the equivalent value in Points is 36. */
    GVLI_increment = MakeWWFixed(36.0);
}
```

For example, if your display format is GVDF_INCHES and you wish to set an initial value of 1 inch and an increment of 1/2 an inch, you should set these values to 72 (points which equals 1 inch) and 36 (points which equals one-half inch) respectively. This is necessary because the system expects these values to be in Points for other system operations. A conversion table is provided in Table 8-1 for setting up these initial values.

Table 8-1 *Conversions to US Points*

Distance Unit	Multiplier
Inches	72
Centimeters	28.3465
Millimeters	2.8346
Picas	12
European Points	1.0656
Ciceros	12.7872

Any increments for GVDF_POINTS_OR_MILLIMETERS or GVDF_INCHES_OR_CENTIMETERS are assumed to be in US units. If the application instead operates in metric, the increment will be automatically converted to a metric equivalent by the system; this metric equivalent will usually be rounded to a convenient numerical value. You may specify an ATTR_GEN_VALUE_METRIC_EQUIVALENT to override the default increment that the system calculates, however.

For example, assume GVDF_INCHES_OR_CENTIMETERS is selected and the value and increment are 72 (points which equals 1 inch). If the application is in US units, the display will specify inches and the value and increment will be 72 (1 inch); if the application is instead in metric, the display will specify centimeters and the increment will be 70.866 (2.5 cm). If instead, you choose an ATTR_GEN_VALUE_METRIC_INCREMENT of 52.692 (2 cm), that will be the increment used if the application is metric.

Code Display 8-7 ATTR_GEN_VALUE_METRIC_INCREMENT

```
/* If the application is US, the initial value will be 1 inch and the increment
 * will be 1 inch. If the application is metric, the initial value will be 2.54 cm
 * (1 inch or 72 points) but the increment will be 2.0 cm (56.692 points). If
 * ATTR_GEN_VALUE_METRIC_INCREMENT were not included, the system would have chosen
 * an increment of 2.5 cm (70.866 points) which is the closest "nice" value to the
 * original increment of 72 points (1 inch). */

@object GenValueClass MyValue = {
    GI_visMoniker = "My Value";
    GVLI_displayFormat = GVDF_INCHES_OR_CENTIMETERS;
    GVLI_value = MakeWWFixed(72.0);
    GVLI_increment = MakeWWFixed(72.0);
    ATTR_GEN_VALUE_METRIC_INCREMENT = MakeWWFixed(56.692);
}
```

To set a new display format, send MSG_GEN_VALUE_SET_DISPLAY_FORMAT. To retrieve the current display format, send the GenValue MSG_GEN_VALUE_GET_DISPLAY_FORMAT. Note that changing the display format will not change the numerical value of that display. For example, if the display format changes from decimal to integer, the GenValue will round the number down and display only the integer portion of the value.

■ MSG_GEN_VALUE_GET_DISPLAY_FORMAT

GenValueDisplayFormat MSG_GEN_VALUE_GET_DISPLAY_FORMAT();

This message returns the *GVLI_displayFormat* field of the GenValue.

Source: Unrestricted.

Destination: Any GenValue object.

Parameters: None.

Return: **GenValueDisplayFormat** of the GenValue.

Interception: Generally not intercepted.

■ MSG_GEN_VALUE_SET_DISPLAY_FORMAT

```
void        MSG_GEN_VALUE_SET_DISPLAY_FORMAT(
            GenValueDisplayFormat   format);
```

This message sets the *GVLI_displayFormat* of the GenValue to the given format. The current value in *GVLI_value* is unaffected, but the display will be updated to reflect the new display format.

Source: Unrestricted. This message is also used internally when the GenValue is being built.

Destination: Any GenValue object.

Parameters: *format* **GenValueDisplayFormat** to set the GenValue to.

Return: Nothing.

Interception: Generally not intercepted.

8.3.6 Sending an Action

```
GVLI_applyMsg, GVLI_destination,
HINT_VALUE_NAVIGATE_TO_NEXT_FIELD_ON_RETURN_PRESS,
HINT_VALUE_CUSTOM_RETURN_PRESS,
MSG_GEN_VALUE_GET_APPLY_MSG, MSG_GEN_VALUE_SET_APPLY_MSG,
MSG_GEN_VALUE_GET_DESTINATION,
MSG_GEN_VALUE_SET_DESTINATION
```

GVLI_applyMsg sets the message for the GenValue to send out whenever it has been modified and needs to apply its changes. Whenever a GenValue receives MSG_GEN_APPLY, it will check whether its GVSF_MODIFIED flag has been set; if it has, it will send out its apply message. If a GenValue is operating in immediate mode, these actions will happen immediately, resulting in an immediate action.

GVLI_destination specifies the destination object (or process) to send the *GVLI_applyMsg* to. (This may also be a **TravelOption**, such as TO_APP_TARGET.)

Code Display 8-8 Sending an Apply Message

```
@object GenValueClass MyValue = {
    GI_visMoniker = “My Value”;
    GVLI_value = MakeWWFixed(1.0);
    GVLI_applyMsg = MSG_MY_VALUE_DOUBLE_VALUE;
    GVLI_destination = process;
}

/* Retrieve the current value. This value will be a fixed point dword. */

@method MyValueProcessClass, MSG_MY_VALUE_DOUBLE_VALUE {
    WWFixedAsDWordcurValue;

    curValue = @call MyValue::MSG_GEN_VALUE_GET_VALUE();
    curValue = curValue*2;
    @call MyValue::MSG_GEN_VALUE_SET_VALUE(curValue, 0);
}
```

A GenValue’s changes are typically applied when the user hits the Return key and the GenValue has the focus. You can change this behavior, though, with the following hints: HINT_VALUE_CUSTOM_RETURN_PRESS allows a textually-oriented GenValue to send the specified message to the destination object when the Return key is pressed. HINT_VALUE_NAVIGATE_TO_NEXT_FIELD_ON_RETURN_PRESS instructs the GenValue to navigate (via the UI) to the next textually-activated object (as the tab key works in many situations).

To change a GenValue’s apply message or destination, send it MSG_GEN_VALUE_SET_APPLY_MSG or MSG_GEN_VALUE_SET_DESTINATION, respectively. Use MSG_GEN_VALUE_GET_APPLY_MSG or MSG_GEN_VALUE_GET_DESTINATION to return the current apply message or destination.

The apply message should be defined on the prototype GEN_VALUE_APPLY_MSG, whose values are shown below.

■ GEN_VALUE_APPLY_MSG

```
void      GEN_VALUE_APPLY_MSG(
          WWFixedAsDWord      value,
          word                stateFlags);
```

This prototype defines the message sent out when the GenValue is "applied." The output of the GenValue should handle a message with these parameters.

Source: GenValue, when "applied."

Destination: The GenValue's output (*GVLI_destination*) object.

Parameters: *value* The current value of the GenValue.

stateFlags The **GenValueStateFlags** stored in *GVLI_stateFlags*.

Return: Nothing.

Interception: The destination object should handle the apply message with this format.

■ MSG_GEN_VALUE_GET_APPLY_MSG

```
Message   MSG_GEN_VALUE_GET_APPLY_MSG();
```

This message returns the GenValue's *GVLI_applyMsg*.

Source: Unrestricted.

Destination: Any GenValue object.

Parameters: None.

Return: The apply message of the GenValue.

Interception: Generally not intercepted.

■ MSG_GEN_VALUE_SET_APPLY_MSG

```
void      MSG_GEN_VALUE_SET_APPLY_MSG(
          Message             message);
```

This message sets the apply message (in *GVLI_applyMsg*) for a GenValue.

Source: Unrestricted.

Destination: Any GenValue object.

Parameters: *message* The apply message to set for the GenValue.

Return: Nothing.

Interception: Generally not intercepted.

◆Objects

■ MSG_GEN_VALUE_GET_DESTINATION

```
optr    MSG_GEN_VALUE_GET_DESTINATION();
```

This message returns the current destination object (or process) that the GenValue sends its apply messages to.

Source: Unrestricted.

Destination: Any GenValue object.

Parameters: None.

Return: The destination optr (*GVLI_destination*) of the GenValue.

Interception: Generally not intercepted.

■ MSG_GEN_VALUE_SET_DESTINATION

```
void    MSG_GEN_VALUE_SET_DESTINATION(
        optr   dest);
```

This message sets the *GVLI_destination* field of the range to the passed optr. The object can be a pointer to a specific object in the system (i.e. the GenProcess object) or can be a pointer to a generic location in the system (i.e. a **TravelOption**).

Source: Unrestricted.

Destination: Any GenValue object.

Parameters: *dest* The optr of the new destination object.

Return: Nothing.

Interception: Generally not intercepted.

8.4 Supplemental Usage

Besides altering instance data, there are several other messages and mechanisms for your use in **GenValueClass**.

8.4.1 Adjusting the Value Indirectly

MSG_GEN_VALUE_INCREMENT, MSG_GEN_VALUE_DECREMENT, MSG_GEN_VALUE_SET_VALUE_TO_MINIMUM, MSG_GEN_VALUE_SET_VALUE_TO_MAXIMUM

To increase the value of *GVLI_value* by the increment in *GVLI_increment*, send the GenValue MSG_GEN_VALUE_INCREMENT. To decrease the value of *GVLI_value* by the increment in *GVLI_increment*, send the GenValue MSG_GEN_VALUE_DECREMENT. These messages are equivalent to clicking on the UI controls provided to increment or decrement the GenValue (usually up and down arrows). Both of these messages clear the indeterminate state of the object and *do not change* its modified state.

To set the value of *GVLI_value* to the minimum in *GVLI_minimum*, send the GenValue MSG_GEN_VALUE_SET_VALUE_TO_MINIMUM. To set the value of *GVLI_value* to the maximum in *GVLI_maximum*, send the GenValue MSG_GEN_VALUE_SET_VALUE_TO_MAXIMUM.Both of these messages clear the indeterminate state of the GenValue and *do not change* its modified state.

Note that HINT_VALUE_NOT_INCREMENTABLE has no effect on these messages; that hint only removes any user controls for incrementing or decrementing a value.

■ MSG_GEN_VALUE_INCREMENT

```
void      MSG_GEN_VALUE_INCREMENT();
```

This message increases the value of *GVLI_value* within the GenValue by the increment in *GVLI_increment*. This message will clear the indeterminate flag of the GenValue but will not affect its modified flag.

Source: Unrestricted. This message is also used internally when responding to user actions.

Destination: Any GenValue object.

Parameters: None.

Return: Nothing. *GVLI_value* will be incremented (or set to the maximum if increasing the value would push it over the maximum).

Interception: Generally not intercepted.

■ MSG_GEN_VALUE_DECREMENT

```
void        MSG_GEN_VALUE_DECREMENT();
```

This message decreases the value of *GVLI_value* within the GenValue by the increment in *GVLI_increment*. This message will clear the indeterminate flag of the GenValue but will not affect its modified flag.

Source: Unrestricted. This message is also used internally when responding to user actions.

Destination: Any GenValue object.

Parameters: None.

Return: Nothing. *GVLI_value* will be decremented (or set to the minimum if decreasing the value would push it below the minimum).

Interception: Generally not intercepted.

■ MSG_GEN_VALUE_SET_VALUE_TO_MINIMUM

```
void        MSG_GEN_VALUE_SET_VALUE_TO_MINIMUM();
```

This message sets the value of *GVLI_value* to the minimum value in *GVLI_minimum*. This message will clear the indeterminate flag of the GenValue but will not affect its modified flag.

Source: Unrestricted.

Destination: Any GenValue object.

Parameters: None.

Return: Nothing. *GVLI_value* will be set to *GVLI_minimum*.

Interception: Generally not intercepted.

■ MSG_GEN_VALUE_SET_VALUE_TO_MAXIMUM

```
void        MSG_GEN_VALUE_SET_VALUE_TO_MAXIMUM();
```

This message sets the value of *GVLI_value* to the maximum value in *GVLI_maximum*. This message will clear the indeterminate flag of the GenValue but will not affect its modified flag.

Source: Unrestricted.

Destination: Any GenValue object.

Parameters: None.

Return: Nothing. *GVLI_value* will be set to *GVLI_maximum*.

Interception: Generally not intercepted.

8.4.2 Status Messages

`ATTR_GEN_VALUE_STATUS_MSG, MSG_GEN_VALUE_SEND_STATUS_MSG`

If your GenValue is operating in delayed mode, there usually occur times when your GenValue's state may not reflect the most recent changes. In most cases this is fine, but in some cases you may wish other UI objects to be notified of a change in your GenValue's state *without* sending out an apply message. This can be done with a status message.

A status message allows your GenValue to send out a message whenever the user interacts with the GenValue, regardless of whether that change will be immediately applied. This is most useful for cases in which two UI objects are representing information that depends on each other. The status message allows one UI object to inform its friend that its state has changed, and that the friend should change its state to reflect the new information.

To give a GenValue a status message, include ATTR_GEN_VALUE_STATUS_MSG in the object's declaration. Use the prototype GEN_VALUE_STATUS_MSG to define your status message. This prototype ensures that the status message passes the correct parameters (the current value and state flags of the GenValue).

Any user changes that do not result in the sending of the object's apply message will result in the sending of the object's status message. For an object in immediate mode, this ATTR will have no effect. You may also manually send an object's status message by sending the GenValue MSG_GEN_VALUE_SEND_STATUS_MSG.

■ GEN_VALUE_STATUS_MSG

```
void      GEN_VALUE_STATUS_MSG(
          WWFixedAsDWord      value,
          word                stateFlags);
```

This prototype should be used to define the status message of the GenValue.

Source: The GenValue, when its status message is sent.

Destination: The GenValue's destination object (*GVLI_destination*).

◆ Objects

Parameters: value — The current user value of the GenValue.

stateFlags — The current value of *GVLI_stateFlags*.

Return: Nothing.

Interception: Must be handled by the output object if the status message is to have any effect.

■ MSG_GEN_VALUE_SEND_STATUS_MSG

```
void    MSG_GEN_VALUE_SEND_STATUS_MSG(
        Boolean            modifiedState);
```

This message sends the status message stored in the object's ATTR_GEN_VALUE_STATUS_MSG instance field. You should pass this message the modified State you wish to send. This modified state may or not reflect the GVSF_MODIFIED flag in the GenValue's *GVLI_stateFlags*.

Source: Unrestricted.

Destination: Any GenValue object.

Parameters: *modifiedState* — TRUE if this message should pass the modified bit (GVSF_MODIFIED) set, FALSE if it should pass GVSF_MODIFIED cleared.

Return: Nothing.

Interception: Generally not intercepted.

8.4.3 Retrieving Text

```
MSG_GEN_VALUE_GET_VALUE_TEXT,
MSG_GEN_VALUE_SET_VALUE_FROM_TEXT,
MSG_GEN_VALUE_GET_MAX_TEXT_LEN,
MSG_GEN_VALUE_SET_MAX_TEXT_LEN, MSG_GEN_VALUE_SELECT_TEXT
```

The GenValue's numeric values are displayed within a special text field. In addition to retrieving the numeric value of the GenValue, you may also retrieve the textual representation of that number with MSG_GEN_VALUE_GET_VALUE_TEXT. Similarly, you can set the value of the GenValue from a textual representation with MSG_GEN_VALUE_SET_VALUE_FROM_TEXT.

You must pass these messages a **GenValueType** which specifies the instance data field you are wishing to set or get. The **GenValueType** types are:

```
typedef enum /* word */ {
      GVT_VALUE,         /* GVLI_value */
      GVT_MINIMUM,       /* GVLI_minimum */
      GVT_MAXIMUM,       /* GVLI_maximum */
      GVT_INCREMENT,     /* GVLI_increment */
      GVT_LONG,          /* Longest value we can
                          * create. */
      GVT_RANGE_LENGTH,  /* End of the displayed
                          * range, if applicable. */
      GVT_RANGE_END,     /* Last value in the range,
                          * if applicable. */
      GVT_VALUE_AS_RATIO_OF_AVAILABLE_RANGE
                         /* The current value,
                          * relative to minimum. */
} GenValueType;
```

Some special subclasses of the GenValue will want to calculate how many characters it will allow the user to type into its text field. By default, the number of characters is determined by the maximum and minimum values, but subclasses can handle MSG_GEN_VALUE_GET_MAX_TEXT_LEN to set it specifically. This message is sent by the range to itself when deciding how big the text field should be. The maximum text length allowed is thirty characters long.

If you wish to select a GenValue's text, send the GenValue MSG_GEN_VALUE_SELECT_TEXT. The specific UI has final say on whether it allows a GenValue to exhibit selectable text.

■ MSG_GEN_VALUE_GET_VALUE_TEXT

```
void     MSG_GEN_VALUE_GET_VALUE_TEXT(
         char                 *buffer,
         GenValueType         valueType);
```

This message retrieves a fixed point value (either *GVLI_value*, *GVLI_minimum*, *GVLI_maximum*, or *GVLI_increment*) from the GenValue and stores its textual representation (in a null-terminated text string) in the passed buffer. This message is not affected by the indeterminate state of the GenValue.

Source: Unrestricted.

Destination: Any GenValue object.

Parameters: *buffer* The pointer to the buffer to store the null-terminated text string. This buffer must be at least GEN_VALUE_MAX_TEXT_LEN bytes long.

valueType The **GenValueType**, specifying the instance data to retrieve and convert into text.

Return: Nothing. The *buffer* will be filled in with text.

Interception: Can be intercepted by a subclass of GenValue to allow custom text formats to be displayed on the screen. In this case you would also subclass MSG_GEN_VALUE_SET_VALUE_FROM_TEXT.

■ MSG_GEN_VALUE_SET_VALUE_FROM_TEXT

```
void    MSG_GEN_VALUE_SET_VALUE_FROM_TEXT(
        char                *text,
        GenValueType        valueType);
```

This message sets a fixed point value (either *GVLI_value*, *GVLI_minimum*, *GVLI_maximum*, or *GVLI_increment*) of the GenValue from a textual representation (in a null-terminated text string) in the passed buffer. This message clears the indeterminate state of the GenValue but does not change its modified state.

Source: Unrestricted.

Destination: Any GenValue object.

Parameters: *text* The pointer to a null-terminated text string to set the GenValue's text to.

valueType The **GenValueType**, specifying the instance data to convert into a fixed point value from the passed text.

Return: Nothing. The instance field will be updated.

Interception: Can be intercepted by a subclass of GenValue to allow custom text formats to be displayed on the screen. In this case, you would also subclass MSG_GEN_VALUE_GET_VALUE_TEXT.

■ MSG_GEN_VALUE_GET_TEXT_LEN

```
byte    MSG_GEN_VALUE_GET_TEXT_LEN();
```

This message may be used by subclasses to determine the number of characters the user is allowed to type into the GenValue's text field.

Source: The GenValue object, when calculating its size.

Destination: Sent to self.

Parameters: None.

Return: The number of characters of text the user will be allowed to type into the GenValue's text field.

Interception: Subclasses may intercept this message to return a specific number of characters; there is no need to call the superclass.

■ MSG_GEN_VALUE_GET_MAX_TEXT_LEN

```
byte    MSG_GEN_VALUE_GET_MAX_TEXT_LEN();
```

This message retrieves the maximum number of characters allowed to be typed into a textual GenValue. By default, this number is calculated from the minimums and maximums specified. Subclasses of GenValue may wish to intercept this message to allow different text lengths (such as values represented with floating point or enumerated type instance data).

Source: The GenValue object sends this to itself when it needs to figure out its maximum text length.

Destination: Itself.

Return: Maximum number of allowable characters

Interception: May be intercepted to allow different text lengths.

■ MSG_GEN_VALUE_SELECT_TEXT

```
void    MSG_GEN_VALUE_SELECT_TEXT();
```

This message selects a GenValue's text, if allowed by the specific UI.

Source: Unrestricted.

Destination: Any GenValue object.

Interception: Generally not intercepted.

8.4.4 Using Value Ratios

MSG_GEN_VALUE_GET_VALUE_RATIO,
MSG_GEN_VALUE_SET_VALUE_FROM_RATIO

Instead of setting or getting a specific value, you may want in some cases to set or get a value as a ratio; this ratio is determined as the percentage of the difference between the minimum and maximum. For example, if the minimum of a GenValue is 10 and the maximum is 100, the 50% ratio would be 55 (halfway between 10 and 100).

To retrieve the value of a GenValue as a ratio, send it MSG_GEN_VALUE_GET_VALUE_RATIO. This message will return the ratio as a dword value. You must also pass this message the **GenValueType** telling it which instance data you wish to retrieve.

To set a value within a GenValue as a ratio, send it MSG_GEN_VALUE_SET_VALUE_RATIO, passing it the ratio and the **GenValueType** (instance data field) to set. The correct value will be computed and set within your GenValue's instance data.

■ MSG_GEN_VALUE_GET_VALUE_RATIO

```
WWFixedAsDword MSG_GEN_VALUE_GET_VALUE_RATIO(
        GenValueType        valueType);
```

This message gets the value of a GenValue (*GVLI_value*) as a ratio of its distance between the minimum value and the maximum value. It returns this ratio as a dword (0000.0000h meaning 0%, ffff.ffffh meaning 100%).

Source: Unrestricted.

Destination: Any GenValue object.

Parameters: *valueType* **GenValueType** of the instance data field to get the value ratio for.

Return: The ratio as a dword.

Interception: You may intercept subclasses of **GenValueClass** to allow custom text formats to be displayed on the screen. Someone subclassing this message for this reason would also want to subclass MSG_GEN_VALUE_SET_VALUE_FROM_TEXT to correctly parse text to yield a new value.

■ MSG_GEN_VALUE_SET_VALUE_RATIO

```
void      MSG_GEN_VALUE_SET_VALUE_RATIO(
          WWFixed            ratio,
          GenValueType       valueType);
```

This message sets the value (*GVLI_value*) of a GenValue as a ratio of the distance between its minimum and the maximum values. This ratio should be in the form of a dword (0000.0000h meaning 0%, ffff.ffffh meaning 100%).

Source: Unrestricted.

Destination: Any GenValue object.

Parameters: *ratio* The dword ratio value.

valueType The **GenValueType** of the instance data to set as a ratio.

Return: Nothing.

Interception: Generally not intercepted.

8.4.5 Text Filters for the GenValue

MSG_GEN_VALUE_GET_TEXT_FILTER

One of the GenValue's components is a text field to enter values into. When this visual component is being built, the GenValue will send MSG_GEN_VALUE_GET_TEXT_FILTER to itself to set up a text filter. A text filter allows a textual object to accept or reject certain characters on a number of bases. The default **VisTextFilter** provides numeric-only filtering with no spaces and no tabs on GVDF_INTEGER or GVDF_DECIMAL display formats; no tabs on distance unit display formats.

You may set up your own filter by intercepting this message and returning a **VisTextFilter** of your own choosing.

■ MSG_GEN_VALUE_GET_TEXT_FILTER

```
VisTextFilter MSG_GEN_VALUE_GET_TEXT_FILTER();
```

This message retrieves the text filtering in use on a GenValue object. By default, GenValues use numeric-only filtering with no spaces and no tabs for numbers and a filtering of no tabs for distance units.

Source: Unrestricted. This message is normally sent by a GenValue to itself when building its textual component.

Destination: Any GenValue object.

Parameters: None.

Return: **VisTextFilter** in use by the textual portion of the GenValue.

Interception: Usually, you will want to intercept this message if you subclass **GenValueClass** and provide your own custom filtering.

8.4.6 Using Ranges in GenValues

```
HINT_VALUE_DISPLAYS_RANGE,
MSG_GEN_VALUE_SET_RANGE_LENGTH,
MSG_GEN_VALUE_GET_RANGE_LENGTH,
MSG_GEN_VALUE_ADD_RANGE_LENGTH,
MSG_GEN_VALUE_SUBTRACT_RANGE_LENGTH
```

Special GenValue objects may actually display ranges rather than individual values within a range—for example, the GEOS scrollbars in OSF/Motif display the percentage of the document visible in the view.

For your GenValue to display a range, use HINT_VALUE_DISPLAYS_RANGE. If this hint is not present, the GenValue is assumed to have a range size of zero. This hint takes an argument of **WWFixedAsDWord** to indicate the size of the range.

To get the range length, use MSG_GEN_VALUE_GET_RANGE_LENGTH. To set the range, use MSG_GEN_VALUE_SET_RANGE_LENGTH. Two other messages, MSG_GEN_VALUE_ADD_RANGE_LENGTH and MSG_GEN_VALUE_SUBTRACT_RANGE_LENGTH, add or subtract the value of the GenValue by the range length.

■ MSG_GEN_VALUE_GET_RANGE_LENGTH

```
WWFixedAsDWord MSG_GEN_VALUE_GET_RANGE_LENGTH();
```

This message returns the value stored in HINT_VALUE_DISPLAYS_RANGE.

Source: Unrestricted.

Destination: Any GenValue object.

Parameters: None.

Return: The value stored in HINT_VALUE_DISPLAYS_RANGE. If this hint is not set, the return value will be zero.

Interception: Generally not intercepted.

■ MSG_GEN_VALUE_SET_RANGE_LENGTH

```
void        MSG_GEN_VALUE_SET_RANGE_LENGTH(
            WWFixedAsDWord value);
```

8.4

This message has the effect of adding or changing the hint HINT_VALUE_DISPLAYS_RANGE for the GenValue. Setting a value of zero will cause the GenValue to act as if it did not have this hint.

Source: Unrestricted.

Destination: Any GenValue object.

Parameters: *value* The new range length.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_VALUE_ADD_RANGE_LENGTH

```
void        MSG_GEN_VALUE_ADD_RANGE_LENGTH();
```

This message adds the range length to the current value of the GenValue object. It has the effect of incrementing the *GVLI_value* field by the range length.

Source: Unrestricted.

Destination: Any GenValue object.

Interception: Generally not intercepted.

■ MSG_GEN_VALUE_SUBTRACT_RANGE_LENGTH

```
void        MSG_GEN_VALUE_SUBTRACT_RANGE_LENGTH();
```

This message subtracts the range length from the current value of the GenValue object. It has the effect of decrementing the *GVLI_value* field by the range length.

Source: Unrestricted.

Destination: Any GenValue object.

Interception: Generally not intercepted.

GenView

9

The GenView provides a means of displaying data on the screen in a scrollable, sizeable, scalable window. This data may be in the form of a graphics document, a generic object tree, or a visible object tree. If your geode requires a means of displaying data visually, you will most likely want to use a view.

This chapter discusses how to create and use the GenView. You should have a good knowledge of the UI in general, and you should understand the GEOS coordinate system.

9.1 GenView Overview

The primary goal of the GenView is to display data on the screen. It provides a window into which applications may draw any type of document from a generic object tree to a visible object tree to graphics strings defined by graphics routines. Nearly all document-oriented applications will use a view in some capacity.

9.1.1 GenView Model

The GenView is simply an object that maintains a window for your geode's use. It must be instructed in how to display your data, although it defaults to a certain display scheme. It receives and sends out messages that define how and when your data is drawn; you must provide an object that can receive the view's messages and respond correctly.

The object that communicates with the view on the application's side is known as the *content* object. The content may be one of three different classes (see below) depending on the type of data you wish to display. Essentially, what type of object you designate as your content is irrelevant to the view; the view sends and receives the same messages regardless.

There are three basic models you may employ based on the type of data you have to display:

- Graphic document
  If you are just displaying standard graphics, you will want the content to be an object of **ProcessClass**. This means that your application's Process object will receive all the messages sent by the view (including MSG_META_EXPOSED, described later) and must know how to draw the graphics document when notified. This model is common.

- Visible object tree
  If you have a tree of visible objects, you will want to set the view's content as an object of **VisContentClass**. This object understands how to respond to several of the view's messages and how to draw a visible tree when notified. This approach is also common.

- Generic object tree
  Though seldom used, it is possible to display generic gadgets in a scrolling view by setting the view's content to be a GenContent object. The GenContent is analogous to the VisContent but works with generic object trees rather than visible object trees. The GenContent is used most often as the superclass of **GenDocumentClass** (see "GenDocument," Chapter 13).

As described in "Graphics Environment," Chapter 23 of the Concepts Book, GEOS uses two different coordinate systems, both of which are based on points (1/72 inch). The default system uses 16-bit coordinates and allows documents up to 113 inches on a side. The large model uses 32-bit coordinates, resulting in documents up to 2,147,483,648 points on a side, or 940 miles square. The GenView uses the large model (32-bit coordinates) by default. This means that all coordinates related to the view will be 32 bits, though you can still use 16 bit coordinates for your documents.

Because the graphic document model described above is the simplest, this chapter will focus on having a **ProcessClass** object as your view's content.

9.1.2 View Features and Goals

The GenView is a powerful and flexible object that does most of your display work for you. It is designed to provide not only the functionality required by nearly all applications but also many additional features that more advanced applications will use. The following goals helped shape the GenView:

- Common functionality
  The view, like all of GEOS, is designed with common application functionality in mind. Those functions common to many applications are integrated as much as possible to relieve programmers of having to code and debug the same features again and again.

- Consistency of use
  The view, like every generic UI object, allows all GEOS applications to behave and look consistent. Users find this important when using several different applications.

- Ease of use
  The view, in its simplest form, is almost trivial to implement. Applications can gain a scrolling window by adding very little code (as evidenced in the Hello World program).

- Expandability
  Using the view's more advanced features requires little more than incremental changes to an application; you can easily and quickly add scaling to or change the background color of your view.

To meet these goals, the view provides several advanced features and makes them easy to use for all applications. All the following features are available to applications with limited effort, and most are expandable to be very powerful.

- Drawing to the view
  Drawing to the view is simple, as simple as calling the proper graphics routines to draw to a GState. The view will send a MSG_META_EXPOSED to its content object whenever the document must be drawn (for example, when the view scrolls or when a window has uncovered a portion of the view); the content then simply must draw it. The view will handle all necessary clipping, placement, scaling, and translation for the video driver. See section 9.4.2 on page 527.

- Scrolling
  You can set up your view to be scrollable and all scrolling to be automatic. You can also cause the view to scroll or even change how the view scrolls. See section 9.4.7 on page 543.

- Scaling
  Every view has a scale factor that determines the scale at which the document is drawn. You may leave this at the default (the normal size), or you may dynamically change it. The view simply asks the content

object to redraw its document, and the view will handle all the scaling geometry. See section 9.4.5 on page 538.

- Sizing
Because most Views are resizable, you can easily alter the sizing behavior of your view. You may also set it to have a certain minimum, maximum, or fixed size. See section 9.4.4 on page 535.

- Setting the Background Color
Depending on the view's purpose, you may wish it to be either the same color as its GenPrimary parent or a different color. The View in Hello World, for example, is white; it could easily be any other color. See section 9.3.3 on page 519.

- Handling Input
Unless your view is display-only, your content will be able to handle either pointer or keyboard events passed through it. You can specify the level of input you want and easily get mouse or typed character events. In addition, the Ink standard for pen input is also supported.

- Setting the Content Size and Origin
Every content object has a size associated with it. The size defines where the content sits in the coordinate space and what part of the coordinate space the view has access to for scrolling. Every view also has an origin describing which part of the content is currently visible. See section 9.4.2 on page 527.

- Changing the Content
Although most Process objects will hang on to their Views, it is possible to change the view's content object. This is most likely used by applications that use the VisContent rather than the Process object as their content. See section 9.4.10 on page 572.

- Defining Children of the view
Every view may have UI objects designated as its children. These objects may appear at any edge of the view. By defining special scroller objects as children of the view, you can alter their placement; for example, you could have the vertical scroller always appear on the left rather than whatever is the default (typically the right). See section 9.4.6 on page 542.

◆Objects

9.1.3 The GenViewControl Object

The GenView, like many objects in the UI, has its own UI controller object to further simplify its use. Though you can implement anything in the controller on your own, it's much easier simply to include a GenViewControl object and let it control the View features for you.

GenViewControlClass is described after the GenView; you must understand the GenView before you can take full advantage of the controller. See section 9.5 on page 574 for complete details.

9.2 Getting Started: View Basics

As stated earlier, this chapter will demonstrate how an application would display a graphic document in a scrollable window. You can look at the Hello World and VisSamp sample applications for examples of the view's features and how to manipulate them. The entire code listing of the Hello World application is located in "First Steps: Hello World," Chapter 4 of the Concepts Book.

This section is meant to show you the basic requirements for implementation of a view. It begins with a review of the concepts employed by the GEOS graphics system and then describes how to define a GenView in your Goc code. It also shows how to handle important messages sent by the view.

9.2.1 Graphics System Review

The GEOS graphics system is explained in full in "Graphics Environment," Chapter 23 of the Concepts Book. Some of it is reviewed here, although you should read the graphics discussions to fully understand how to draw a graphic document.

The GEOS default document space is represented by signed, 16-bit coordinates relative to a central origin. Coordinates are in 72 points per inch, and documents that use this space can be up to 8192 points, or 113.8 inches, on a side. If your document must be larger than this, you may make use of the “large document” model, which uses 32-bit signed coordinates. Large documents can be over 940 miles on a side.

Every document is drawn to a GState which keeps track of such current state as pen position, color, rotation, and scale. You may apply transformations on the GState to get different rotation and placement within the document space.

To display a graphic document in a view, you must set up the view to have your Process object as its content. This means that your Process object must be able to draw the document, handle messages received from the view, and control the view’s behavior as required. The Process object must also know how to support the UI mechanisms with which the view’s content object might want to interact (such as the quick-transfer mechanism).

The view defaults to using the large coordinate space for all its operations, though this does not mean only large documents can be displayed in the view. Large documents are represented in a signed, 32-bit (long integer) document coordinate space. This means large documents may be up to 2,147,483,648 points high by 2,147,483,648 points wide (941.5 miles on a side). Because the graphics system is optimized for 16-bit operations, there are special considerations you must take into account when using a view to display large documents.

There are two basic ways to deal with 32-bit coordinates in a 16-bit graphics system. Both involve performing an extended translation on the GState in order to draw in a “local” 16-bit graphics space. The extended translation is performed with **GrApplyTranslationDWord()**, which takes two 32-bit coordinates and translates the GState’s origin to those coordinates. Any 16-bit drawing operations are then automatically done relative to the new origin. All translations are cumulative.

One of the two ways to handle large documents is to apply the extended translation before each drawing operation and untranslate after each operation. This obviously incurs quite a bit of additional overhead.

The other way of handling large documents is to “tile” them, just as the Spooler does when printing large documents on small paper. In this case, you would apply the extended translation once, from then on drawing in 16-bit coordinates relative to the local origin. When you wanted to move to another tile, you would apply another extended translation to the new tile. The drawback to this is that your drawing code has to recognize when the 16-bit local boundaries are being crossed and a transformation is necessary.

9.2.2 Defining the Basic View

To use a view to display a document, you should define it in your application's Goc code. The view can legally be a child of GenPrimary, GenDisplay, or GenInteraction. In simple applications, the view will be a child of the 9.2
application's GenPrimary object. More complex applications will have multiple displays with a view in one or more GenDisplay objects.

The most basic view is resizable and has a white background. You may draw any size document in this view, and it will be clipped properly by the windowing system. At its most basic, the view is not scrollable; however, by setting two attributes as shown in Code Display 9-1, the view becomes scrollable in both dimensions.

The most common view is resizable and scrollable with a white background. It generally has boundaries on its scrollable area, called document bounds. Code Display 9-1 shows this type of view.

Notice that the Goc code for the view is simple; much of the power of the view as a display tool rests in its dynamic flexibility. Once the basic view is defined, you can change its scale, color, scrolling behavior, and document bounds simply by sending messages to it. This dynamic flexibility is also exploited extensively by the GenViewControl.

Code Display 9-1 The Basic GenView

```
/*
 * The GenView object creates a window in which the application may draw all or
 * portions of its document. Whenever a portion of this window becomes “invalid”
 * (when the window is resized or moved, for example), the view will send
 * MSG_META_EXPOSED to its content (in this case the application’s Process object)
 * indicating that a redraw is required. The view keeps track of the clipping
 * boundaries and will automatically display the document properly.
 */
```

```
/*
 * The view defined here provides a window with the default white background and
 * normal scrolling and sizing behavior. Its document boundaries are set to 11
 * by 17 inches (the default has all bounds set to zero). The view provides
 * scrollbars automatically, and the application does not need to understand how
 * scrolling is implemented; instead, the view will simply request a redraw of the
 * entire document.
 */

@object GenViewClass MyView = {

        /*
         * The document bounds are set to build an 11 x 17 inch document. Document
         * coordinates are in 1/72 inch (points), and document bounds must also be
         * specified in points. The upper-left corner of the document is placed at
         * (0,0), and the coordinates increase down and right to the far corner
         * (792, 1224).
         */

        GVI_docBounds = {
            0,          /* left bound */
            0,          /* top bound */
            792,        /* right bound */
            1224};      /* bottom bound */

        /*
         * The GVI_horizAttrs and GVI_vertAttrs fields determine the view’s
         * scrolling, linking, and sizing behavior in the appropriate dimension.
         * Setting the scrollable attribute in both these fields will make the view
         * create and maintain scrollers (in OSF/Motif, these will appear as
         * scrollbars). All scrolling will happen automatically.
         * These lines are necessary if scrolling is desired.
         */
        GVI_horizAttrs = @default | GVDA_SCROLLABLE;
        GVI_vertAttrs = @default | GVDA_SCROLLABLE;

        /* Lastly, we must designate which object (our Process object in this case)
         * will receive and handle the message to draw the displayed data
         * (MSG_META_EXPOSED). This object is called the content object and may
         * be the Process, a GenContent, or a VisContent.
         * Note that no message is defined here as with most action descriptors;
         * the view will send only MSG_META_EXPOSED. You must set this attribute no
         * matter what object is designated as the content; when using a
         * GenContent or a VisContent, the object's name should appear in
         * place of process. */
        GVI_content = process;
}
```

9.2.3 Handling View Messages

After defining the GenView, you must create the method that will handle MSG_META_EXPOSED sent by the view in order to draw in the view's window. This is absolutely necessary, even if your view is not scrollable; the view will send MSG_META_EXPOSED when any part of its display window becomes invalid—this includes when the view is first instantiated and its window is first displayed.

In this simple model, MSG_META_EXPOSED is the only message your application will need to handle relating to the view. If you are managing large documents, then you may also have to handle a few others pertaining primarily to input. A sample handler for MSG_META_EXPOSED—one you can modify and put in your own code—is shown in Code Display 9-2.

If you are planning on displaying visible objects in the view (as opposed to having your Process object draw graphics), you should instead pay attention to MSG_VIS_DRAW. This message is defined by **VisClass** and is sent by a VisContent object to itself as the default behavior for MSG_META_EXPOSED. For full information on MSG_VIS_DRAW and visible objects, see "Drawing to the Screen" on page 1315, in "VisClass," Chapter 23.

Code Display 9-2 MSG_META_EXPOSED Handler

```
/* Each time some part of the view window becomes invalid, the view will send a
 * MSG_META_EXPOSED to its content object. When MSG_META_EXPOSED is received,
 * the content must draw the document to a newly-created GState, then pass the
 * GState to the windowing system for drawing to the screen. */

/* The format of this message is
 *      void MSG_META_EXPOSED(WindowHandle win);
 * The passed value is the window handle of the view window; it will be used in
 * conjunction with the temporary GState for drawing. */
```

```
@method MyProcessClass, MSG_META_EXPOSED {
    /* Set up a temporary GState handle variable for our document. */
    GStateHandle tempGState;

    /* Initialize the GState to the default. */
    tempGState = GrCreateState(win);

    /* Now start a window update. The routine GrBeginUpdate() notifies the window
     * system that we are in the process of drawing in the region in the window
     * that was invalidated. GrBeginUpdate() allows drawing to be clipped to only
     * the region that was invalidated. */

    GrBeginUpdate(tempGState);

    /* Now that we have a GState and have notified the windowing system that we
     * intend to draw, we can draw our document. Here, your method should call
     * whatever routines are appropriate for drawing the entire document.
     * A good hint here relates also to printing: Because printing and drawing to a
     * view are essentially the same operation, you might want to consolidate all
     * your drawing into a single routine that is called by both this method and
     * your printing method. This is what is done here; the routine called
     * MyDrawingRoutine() draws the document we wish displayed. MyDrawingRoutine()
     * will also be called by our printing method with a different GState. */

    MyDrawingRoutine(tempGState);

    /* Now that we have finished drawing the document, we must inform the window
     * system that we are done. Additionally, we must now destroy the temporary
     * GState we used to draw the document. This is essential as not destroying
     * gstates causes small locked blocks to remain on the heap, eroding system
     * performance quickly and eventually making the system run out of handles. */

    GrEndUpdate(tempGState);
    GrDestroyState(tempGState);
}
```

9.3 Basic View Attributes

The GenView has several instance data fields that it uses to maintain its current status. It also has other attribute fields that determine its scrolling, scaling, and sizing behavior as well as the manner in which it accepts input events. Listed in Code Display 9-3 are all the GenView's instance data fields.

Each is described later in the chapter. Additionally, each is dynamically changeable with the use of messages defined by GenView.

Code Display 9-3 GenView Instance Data

```
/* Default settings are shown in the definitions. Other possible settings are
 * shown in a comment after the definition. */

    @instance PointDWFixed      GVI_origin = {{0, 0}, {0,0}};
    @instance RectDWord         GVI_docBounds = {0, 0, 0, 0};
    @instance PointDWord        GVI_increment = {20, 15};
    @instance PointWWFixed      GVI_scaleFactor = {{0, 1}, {0, 1}};
    @instance ColorQuad         GVI_color = {WHITE, 0, 0, 0};
    @instance GenViewAttrs      GVI_attrs = (GVA_FOCUSABLE);
        /* Possible flags for GVI_attrs:
         *      GVA_CONTROLLED          GVA_GENERIC_CONTENTS
         *      GVA_TRACK_SCROLLING     GVA_DRAG_SCROLLING
         *      GVA_NO_WIN_FRAME        GVA_SAME_COLOR_AS_PARENT_WIN
         *      GVA_VIEW_FOLLOWS_CONTENT_GEOMETRY
         *      GVA_WINDOW_COORDINATE_MOUSE_EVENTS
         *      GVA_DONT_SEND_PTR_EVENTS GVA_DONT_SEND_KBD_RELEASES
         *      GVA_SEND_ALL_KBD_CHARS  GVA_FOCUSABLE
         *      GVA_SCALE_TO_FIT        GVA_ADJUST_FOR_ASPECT_RATIO */

    @instance GenViewDimensionAttrs     GVI_horizAttrs = 0;
    @instance GenViewDimensionAttrs     GVI_vertAttrs = 0;
        /* Possible flags for the dimension attribute records:
         *      GVDA_SCROLLABLE         GVDA_SPLITTABLE
         *      GVDA_TAIL_ORIENTED      GVDA_DONT_DISPLAY_SCROLLBAR
         *      GVDA_NO_LARGER_THAN_CONTENT
         *      GVDA_NO_SMALLER_THAN_CONTENT
         *      GVDA_SIZE_A_MULTIPLE_OF_INCREMENT
         *      GVDA_KEEP_ASPECT_RATIO                  */

    @instance GenViewInkType    GVI_inkType = GVIT_PRESSES_ARE_NOT_INK;
        /* Possible flags for GVI_inkType:
         *      GVIT_PRESSES_ARE_NOT_INK        GVIT_INK_WITH_STANDARD_OVERRIDE
         *      GVIT_PRESSES_ARE_INK            GVIT_QUERY_OUTPUT */

    @instance @optr             GVI_content;
    @instance @optr             GVI_horizLink;
    @instance @optr             GVI_vertLink;

/* The following are the hints and attributes defined for the GenView’s vardata.
 * Because GenViewClass is a subclass of GenClass, it inherits all other generic
 * hints as well. */
```

```
    @vardata void       HINT_VIEW_LEAVE_ROOM_FOR_VERT_SCROLLER;
    @vardata void       HINT_VIEW_LEAVE_ROOM_FOR_HORIZ_SCROLLER;

    @vardata void       HINT_VIEW_IMMEDIATE_DRAG_UPDATES;
    @vardata void       HINT_VIEW_DELAYED_DRAG_UPDATES;
    @vardata void       HINT_VIEW_REMOVE_SCROLLERS_WHEN_NOT_SCROLLABLE;
    @vardata void       HINT_VIEW_SHOW_SCROLLERS_WHEN_NOT_SCROLLABLE;

    @vardata InkDestinationInfoParams ATTR_GEN_VIEW_INK_DESTINATION_INFO;
        @reloc ATTR_GEN_VIEW_INK_DESTINATION_INFO,
                word_offsetof(InkDestinationInfoParams, IDIP_dest), optr;

@vardata void ATTR_GEN_VIEW_DOES_NOT_ACCEPT_TEXT_INPUT;

    @vardata XYSize     ATTR_GEN_VIEW_PAGE_SIZE;
    @vardata void       ATTR_GEN_VIEW_SCALE_TO_FIT_BASED_ON_X;
    @vardata void       ATTR_GEN_VIEW_SCALE_TO_FIT_BOTH_DIMENSIONS;
    @vardata void       ATTR_GEN_VIEW_DO_NOT_WIN_SCROLL;

/* By default, all GenView objects are targetable. */
    @default GI_attrs = @default | GA_TARGETABLE;
```

The attributes shown in Code Display 9-3 are all dynamically changeable; in fact, many (such as the scale factor) become interesting only when set or changed as a feature of the application (as in the scaling features of GeoWrite and GeoDraw). Each attribute, therefore, is described under its appropriate section of this chapter.

9.3.1 The GVI_attrs Attribute

`GVI_attrs, MSG_GEN_VIEW_SET_ATTRS, MSG_GEN_VIEW_GET_ATTRS`

The *GVI_attrs* field, as shown in Code Display 9-3, controls many aspects of the view, including several features related to scrolling, input events, and the content object. It is implemented as a bitfield record, and any or all of its attributes may be set at once.

Generally, the attributes specified in this field are not changed at run-time; if your application needs to change or retrieve them, however, it may with the messages MSG_GEN_VIEW_GET_ATTRS and MSG_GEN_VIEW_SET_ATTRS, both of which are shown below.

The attributes that can be set in *GVI_attrs* are shown in the listing below. The default settings are given in Code Display 9-3.

GVA_CONTROLLED
Set if the GenView is connected to a GenViewControl object and should therefore send out notification as appropriate and add itself to the proper GCN lists.

GVA_GENERIC_CONTENTS
Set if the content is a **GenContentClass** object. If this bit is set, the mouse grab mode and pointer sending mode are set by the specific UI, overriding whatever is set in the instance data.

GVA_TRACK_SCROLLING
Set if track scrolling is desired. If this is set, the view will send scrolling events to the content object. The content can then alter the scrolling behavior and send the event on to the view.

GVA_DRAG_SCROLLING
Set if drag scrolling is desired. If this bit is set, drag scrolling (when the user drags outside the window bounds, the window scrolls in that direction) is implemented by the view. The content does not have to do anything special; drag scrolling is implemented entirely by the view.

GVA_NO_WIN_FRAME
Set if no frame around the subview window is desired. This is set only if the view is meant to be a display view with its borders obscured (e.g. a text display that has no frame).

GVA_SAME_COLOR_AS_PARENT_WIN
Set if the background color of the view should be whatever color the parent window's background is. This is used in nearly every case where GVA_GENERIC_CONTENTS is also set.

GVA_VIEW_FOLLOWS_CONTENT_GEOMETRY
Set if the view should follow the size of the content in all non-scrollable dimensions. This can only be used if the view and the content are running in the same thread and if the content is not the process object.

GVA_WINDOW_COORDINATE_MOUSE_EVENTS
Set if mouse coordinates passed to the content are in window coordinates (relative to the window's origin) rather than document coordinates. If the document bounds are set outside

the 16-bit boundaries of the standard graphics system, then this bit *must* be set. For full information on mouse event handling, see "Input," Chapter 11 of the Concepts Book.

GVA_DONT_SEND_PTR_EVENTS
Set if pointer events should not be sent to the GenView's content. Use this for optimization if pointer events do not need to be sent through the view.

GVA_DONT_SEND_KBD_RELEASES
Set if no keyboard releases should be sent to the content. Applications will set this if they only care about presses, not releases.

GVA_SEND_ALL_KBD_CHARS
Set if the view should pass all keyboard presses and releases to the content before the Specific UI has a chance to interpret them. Normally, the Specific UI will receive keyboard events, filter out the ones it uses (including application-defined accelerators), and then pass the unused events on to the view. This flag will, in essence, override the authority of the Specific UI and therefore is generally highly undesirable. If an application finds a need for this flag, the application is responsible for passing unused characters on to the UI for proper processing. If the unused characters do not get passed on, synchronization problems could result; if an application decides to handle reserved characters, the specific UI's reaction will be unpredictable.

GVA_FOCUSABLE
Set if the view window can gain the input focus. This is set by default and should be set for nearly all views.

GVA_SCALE_TO_FIT
Set if the GenView should operate in "scale to fit" mode. In this mode, the vertical scale factor will automatically be set so the entire document fits vertically within the view window; the horizontal scale factor will then be set based on the vertical. This behavior can be modified by designating a page size (via ATTR_GEN_VIEW_PAGE_SIZE), by changing the scaling to be based on the horizontal (via the attribute ATTR_GEN_VIEW_SCALE_TO_FIT_BASED_ON_X), or by setting ATTR_GEN_VIEW_SCALE_TO_FIT_BOTH_DIMENSIONS.

GVA_ADJUST_FOR_ASPECT_RATIO
Set if the GenView should alter its scale factors appropriate to the aspect ratio of the screen.

■ MSG_GEN_VIEW_SET_ATTRS

```
void        MSG_GEN_VIEW_SET_ATTRS(
            word                attrsToSet,
            word                attrsToClear,
            VisUpdateMode       updateMode);
```

This message changes the view's *GVI_attrs* record to new values.

Source: Unrestricted.

Destination: Any GenView object.

Parameters: *attrsToSet* A record of **GenViewAttrs** to set in *GVI_attrs*.

attrsToClear A record of **GenViewAttrs**; each flag set in this parameter will be cleared in *GVI_attrs*.

updateMode A **VisUpdateMode** indicating when the GenView should be visually updated.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_VIEW_GET_ATTRS

```
word        MSG_GEN_VIEW_GET_ATTRS();
```

This message returns a word representing the current *GVI_attrs* record. To check whether an attribute is set, mask the returned word value against the attribute.

Source: Unrestricted.

Destination: Any GenView object.

Parameters: None.

Return: The current **GenViewAttrs** record in the GenView's *GVI_attrs* field.

Interception: Generally not intercepted.

9.3.2 Dimensional Attributes

```
GVI_horizAttrs, GVI_vertAttrs,
MSG_GEN_VIEW_GET_DIMENSION_ATTRS,
MSG_GEN_VIEW_SET_DIMENSION_ATTRS
```

Together, the *GVI_horizAttrs* and *GVI_vertAttrs* fields are called the "dimension attributes." They determine the characteristics of the view in each dimension—whether it is scrollable or splittable, whether a scrollbar should be displayed, and whether certain rules should be applied to sizing and scaling.

These attributes, once set, are not frequently changed during execution. Both of these attributes are bitfield records. Each has the same set of possible flags, and these flags are listed below (they are bits in the record **GenViewDimensionAttrs**):

GVDA_SCROLLABLE
Set if view is scrollable in the given dimension.

GVDA_SPLITTABLE
Set if the view is splittable in the given dimension. This is not currently implemented but is planned.

GVDA_TAIL_ORIENTED
Set if the document prefers to be displayed at its end. This flag will cause the view to stay at the bottom of the document when the document's length or size is changed—if scrolling to the middle or top, however, the tail orientation will be ignored. Currently, tail orientation is not implemented across threads but may be approximated with track scrolling.

GVDA_DONT_DISPLAY_SCROLLBAR
Set if the scroller for the given dimension should not be displayed even if the dimension is scrollable.

GVDA_NO_LARGER_THAN_CONTENT
Set if the view should never get larger than the content's maximum size in the given dimension (based on the current *GVI_docBounds* settings). By default, there are no restrictions on the size of the view—it is generally dictated by the view's parent window unless a size hint is applied to the view.

GVDA_NO_SMALLER_THAN_CONTENT
Set if the view should always stay large enough to show the entire content in the given dimension (based on the current *GVI_docBounds* settings). By default, there are no restrictions on the size of the view.

GVDA_SIZE_A_MULTIPLE_OF_INCREMENT
Set if the view size should always be rounded down to a multiple of the *GVI_increment* value in the given dimension. If greater control over the sizing behavior is required, a subclass of **GenViewClass** should be used with an altered handler of MSG_GEN_VIEW_CALC_WIN_SIZE.

GVDA_KEEP_ASPECT_RATIO
Set if the aspect ratio implied by the view's initial length and width should be maintained. If set in *GVI_horizAttrs*, the view will calculate the width according to the height. If set in *GVI_vertAttrs*, the view will calculate the height according to the width. This flag is normally used in conjunction with HINT_INITIAL_SIZE. It is an error to set this flag in both dimensions.

If your application must retrieve or set these attributes during execution, it may do so with the messages MSG_GEN_VIEW_GET_DIMENSION_ATTRS and MSG_GEN_VIEW_SET_DIMENSION_ATTRS. Also, to combine and separate the attribute fields, you can use the three macros also shown below.

■ MAKE_SET_CLEAR_ATTRS

```
word      MAKE_SET_CLEAR_ATTRS(setAttrs, clearAttrs);
          byte   setAttrs, clearAttrs;
```

This macro takes two byte values and combines them into a one-word argument. It is used with MSG_GEN_VIEW_SET_DIMENSION_ATTRS, below.

■ MAKE_HORIZ_ATTRS

```
byte      MAKE_HORIZ_ATTRS(val);
          word   val;
```

This macro takes a word-sized value and returns the high byte only (used with MSG_GEN_VIEW_GET_DIMENSION_ATTRS, below, to retrieve the *GVI_horizAttrs* flag from the returned value).

■ MAKE_VERT_ATTRS

```
byte     MAKE_VERT_ATTRS(val);
         word   val;
```

This macro takes a word and returns the low byte only. It should be used with MSG_GEN_VIEW_GET_DIMENSION_ATTRS, below, to extract the *GVI_vertAttrs* flag from the returned value.

9.3

■ MSG_GEN_VIEW_SET_DIMENSION_ATTRS

```
void     MSG_GEN_VIEW_SET_DIMENSION_ATTRS(
         word   horizAttrsToSetClear,
         word   vertAttrsToSetClear,
         VisUpdateMode updateMode);
```

This message changes the *GVI_horizAttrs* and *GVI_vertAttrs* records of the view.

Source: Unrestricted.

Destination: Any GenView object.

Parameters: *horizAttrsToSetClear*
A word consisting of two records of **GenViewDimensionAttrs** flags. The high byte represents the flags to be cleared, and the low byte represents the flags to be set; these flags pertain to the *GVI_horizAttrs* field.

vertAttrsToSetClear
A word consisting of two records of **GenViewDimensionAttrs** flags. The high byte represents the flags to be cleared, and the low byte represents the flags to be set; these flags pertain to the *GVI_vertAttrs* field.

updateMode A **VisUpdateMode** type.

Interception: Nothing.

Warnings: Generally not intercepted.

Tips: The macro MAKE_SET_CLEAR_ATTRS is defined above and combines two byte-sized records into one word-sized record for the Goc preprocessor. Its use is shown in the example below.

```
@call MyView::MSG_GEN_VIEW_SET_DIMENSION_ATTRS(
      MAKE_SET_CLEAR_ATTRS(horizSet, horizClear),
      MAKE_SET_CLEAR_ATTRS(vertSet, vertClear),
      VUM_NOW);
```

■ MSG_GEN_VIEW_GET_DIMENSION_ATTRS

```
word      MSG_GEN_VIEW_GET_DIMENSION_ATTRS();
```

This message returns the GenView's current dimension attribute records.

Source: Unrestricted.

Destination: Any GenView object.

Parameters: None.

Return: A word consisting of two **GenViewDimensionAttrs** records. Retrieve the horizontal attributes (*GVI_horizAttrs*) with the macro MAKE_HORIZ_ATTRS and the vertical attributes (*GVI_vertAttrs*) with the macro MAKE_VERT_ATTRS.

Interception: Generally not intercepted.

9.3.3 Setting the Background Color

GVI_color, MSG_GEN_VIEW_SET_COLOR, MSG_GEN_VIEW_GET_COLOR, GVCD_INDEX, GVCD_RED, GVCD_FLAGS, GVCD_BLUE_AND_GREEN, GVCD_BLUE, GVCD_GREEN

A view can have its background color determined by an RGB value, a grayscale value, a CMY value, or as a GEOS color index.

If a view is managing generic objects or is displaying text, it may be most appropriate for the view to appear with the same background color as its parent Primary window. To do this, set the view's *GVI_attrs* field to the following:

```
GVI_attrs = @default | GVA_SAME_COLOR_AS_PARENT_WIN;
```

Otherwise, you will probably want to set the view to something other than the parent's color. To use a standard color index (16-color EGA set), set the *GVI_color* attribute as follows:

```
GVI_color = {index, 0, 0, 0};
      /* index is a GEOS color index, one
       * of the Color enumerated type. */
```

There is no need to set any of the other color fields if you are using an index. The default background color is determined by the specific UI but is normally C_WHITE. (Color indexes are described in “Graphics Environment,” Chapter 23 of the Concepts Book.)

If you want to use the RGB scheme for setting the background color, you must know the structure used for defining the color. The view uses the structure **ColorQuad** to hold all four of the color data fields. This structure is defined as follows:

```
typedef struct {
    byte          CQ_redOrIndex;
                  /* color index or red value */
    ColorFlag     CQ_info;
    byte          CQ_green;   /* green value */
    byte          CQ_blue;    /* blue value */
} ColorQuad;
```

When using a color index, you only need to set the *CQ_info* field to the proper index and set the flags to zero. When using other types of value, however, you must set the *CQ_info* byte to the type and the other fields to their proper values as shown below:

```
GVI_color = { redVal,         /* red RGB value */
              CF_RGB,         /* RGB flag */
              greenVal,       /* green RGB value */
              blueVal }       /* blue RGB value */
```

The **ColorFlag** record determines how the color will appear on the screen. It record may have any of one the following values:

CF_INDEX This flag indicates the color is a palette index, specified in the *CQ_redOrIndex* field.

CF_GRAY This flag indicates the color will be set with a grayscale value as determined by the other three fields.

CF_SAME This flag indicates the color will not be changed (used for hatch patterns, for example).

CF_CMY This flag indicates the color is set with CMY values.

CF_RGB This flag indicates the colors are specified in RGB values in the other three color fields.

As an alternative, you may retrieve, set, or change the view's background color during execution with the messages MSG_GEN_VIEW_SET_COLOR and MSG_GEN_VIEW_GET_COLOR. These messages are shown below. You may also use the macros referenced below to construct and extract color arguments.

■ MSG_GEN_VIEW_SET_COLOR

```
void    MSG_GEN_VIEW_SET_COLOR(
        byte                indexOrRed,   /* color index or red value */
        ColorFlag           flags,        /* color flags */
        word                greenBlue);   /* green and blue values */
```

This message sets the background color of the view. It will not affect the document or printing. The color may be specified either by standard GEOS color index or by RGB value.

Source: Unrestricted.

Destination: Any GenView object.

Parameters: *indexOrRed* If specifying the color as an index, this is the index value. If specifying as an RGB value, this is the red component of the value.

flags A record of **ColorFlag**. If specifying the color as an RGB value, set this to CF_RGB. If specifying the color as a palette index, set this to CF_INDEX.

greenBlue The green and blue components of the RGB value. If specifying color as an index, set this to zero. If specifying as an RGB value, combine the green and blue components into a single parameter with the macro GVC_GREEN_AND_BLUE.

Return: Nothing.

Interception: Generally not intercepted.

9.3

■ MSG_GEN_VIEW_GET_COLOR

```
dword    MSG_GEN_VIEW_GET_COLOR();
```

This message returns all four fields of the view's background color in a single dword value.

Source: Unrestricted.

Destination: Any GenView object.

Parameters: None.

Return: The four color specifier fields of the GenView's current background color. The individual values may be retrieved with the macros below.

Interception: Generally not intercepted.

■ GVCD_INDEX

```
byte     GVCD_INDEX(val);
         dword  val;
```

This macro extracts the color index byte from the given dword value. It is intended for use when an index is used.

■ GVCD_RED

```
byte     GVCD_RED(val)
         dword  val;
```

This macro extracts the red component byte from the given dword value. It is intended for use with RGB values.

■ GVCD_FLAGS

```
byte     GVCD_FLAGS(val)
         dword  val;
```

This macro extracts the color flags byte from the given dword.

■ GVCD_GREEN

```
byte     GVCD_GREEN(val)
         dword  val;
```

This macro extracts the green component byte from the given dword.

■ GVCD_BLUE

```
byte      GVCD_BLUE(val)
          dword  val;
```

This macro extracts the blue component byte from the given dword.

9.3.4 The GVI_increment Attribute

GVI_increment, MSG_GEN_VIEW_GET_INCREMENT, MSG_GEN_VIEW_SET_INCREMENT

Every view has an increment associated with it; this increment may be used in scrolling and sizing the view window. The increment has both a horizontal and a vertical component which are separate from each other. The increment must be specified in document coordinates (points) and is represented by a signed long number. By default, the horizontal increment is set to 20 and the vertical to 15.

The increment values are stored in the *GVI_increment* field. This field is a structure of type **PointDWord**. This structure consists of two signed long integers, as follows:

```
typedef struct {
      sdword      PD_x;       /* x increment */
      sdword      PD_y;       /* y increment */
} PointDWord;
```

It is easy to set the increment values in your Goc code, thus:

```
GVI_increment = {
         horiz,    /* horizontal increment value */
         vert }    /* vertical increment value */
```

In general, the increment will not change during execution. However, if you find it necessary to do so, you can retrieve and set the increment with MSG_GEN_VIEW_GET_INCREMENT and MSG_GEN_VIEW_SET_INCREMENT.

■ MSG_GEN_VIEW_SET_INCREMENT

```
void    MSG_GEN_VIEW_SET_INCREMENT(@stack
        sdword yIncrement,
        sdword xIncrement);
```

This message sets the horizontal and vertical increment values for the view. If the increment had been used before for scrolling, sizing, or scaling, all the same operations will subsequently use the new increment. The two increment values must be in document coordinates (1/72 inch). 9.4

Source: Unrestricted.

Destination: Any GenView object.

Parameters: *yIncrement* The new vertical increment value.

xIncrement The new horizontal increment value.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_VIEW_GET_INCREMENT

```
void    MSG_GEN_VIEW_GET_INCREMENT(
        PointDWord *increment);
```

This message returns the view's current horizontal and vertical increments in document coordinates.

Source: Unrestricted.

Destination: Any GenView object.

Parameters: *increment* A pointer to an empty **PointDWord** structure.

Return: The *increment* pointer will point to the **PointDWord** structure containing the horizontal and vertical increment values.

Interception: Generally not intercepted.

9.4 Advanced Concepts and Uses

The GenView's greatest strength is its dynamic flexibility. By sending a message, you can change the scale factor or scroll to any point in a document. You can even define your own scroller objects and objects that can appear at the edges of the view with the scrollers.

Objects

This section of this chapter explains the features, messages, and functions of the view that beginners can defer for now; however, the view in general is not complicated and can easily be shaped into what you need.

9.4.1 The Life of a View

When a view is first instantiated, the window system wants to make sure everything happens properly. As a result, the view sends out a series of messages intended to notify the content that the view is opening and that it should be ready to draw its document soon. Most of these messages are intended for the GenContent and VisContent objects, and you will not need to handle them if you designate your Process object as the view's content. Similarly, when the view is shutting down, it notifies its content of the fact that it is closing. Again, the Process does not typically need to handle these messages.

9.4.1.1 Handling View Messages

```
MSG_META_CONTENT_SET_VIEW,
MSG_META_CONTENT_VIEW_ORIGIN_CHANGED,
MSG_META_CONTENT_VIEW_SCALE_FACTOR_CHANGED,
MSG_META_CONTENT_VIEW_WIN_OPENED,
MSG_META_CONTENT_VIEW_OPENING,
MSG_META_CONTENT_VIEW_SIZE_CHANGED
```

When a GenView is instantiated, the UI takes care of most of the work. However, the view will send several messages to its new content in order to notify it that the view is opening.

The first message sent will be a MSG_META_CONTENT_SET_VIEW, which you may intercept if you need to know the view's object pointer (optr).

The content will then receive a series of other messages indicating the view's initial state. These messages are also sent whenever the view changes its state—for example, when the user moves a scrollbar, the view's origin will change. In general, your Process will not need to handle these. However, if you are using one of the content objects, you may need them. They are all detailed below (for full reference information on these messages, see

“Messages Received from the View” on page 1443 of “VisContent,” Chapter 25):

See VisContentClass

The messages described here are very important for applications that want to do special view operations. See VisContentClass for full descriptions.

- MSG_META_CONTENT_VIEW_ORIGIN_CHANGED
  This message indicates the initial origin of the view. See section 9.4.2.2 on page 528 for more information.

- MSG_META_CONTENT_VIEW_SCALE_FACTOR_CHANGED
  This message indicates the original scale factor of the view. If you are displaying large documents, you will need to know this in order to handle input events.

- MSG_META_CONTENT_VIEW_WIN_OPENED
  This message indicates that the view window is being created. Unless you plan on keeping track of several open views, you will not need to handle this message. It is sent before the window is actually opened; this means that the view may or may not be mapped, and you should not draw your document in response to this message. You should instead wait until you receive a MSG_META_EXPOSED. Some applications may intercept this message to get the window handle of the GenView and cache it for future use (until the window is closed).

- MSG_META_CONTENT_VIEW_OPENING
  This message indicates that the view has received a MSG_VIS_OPEN and that the content will soon be put on the screen. Your Process object will not likely handle this message, though some special VisContents may in order to prepare information needed before drawing.

- MSG_META_CONTENT_VIEW_SIZE_CHANGED
  This message indicates a change in size of the view—your Process object will not need to intercept this message unless you plan on using the view’s width and height for something special.

After the view has been created and opened and is on the screen, it will send a MSG_META_EXPOSED to the content. At this time, the content must make sure the document gets drawn properly (see section 9.4.3 on page 534). No special handler is required when a view starts up; the MSG_META_EXPOSED is the same as would be received normally.

9.4.1.2 When the View Shuts Down

```
MSG_META_CONTENT_VIEW_CLOSING,
MSG_META_CONTENT_VIEW_WIN_CLOSED,
MSG_META_CONTENT_SET_VIEW
```

See VisContentClass

The messages described here are very important for applications that want to do special view operations. See VisContentClass for full descriptions.

When a view is about to be closed, a MSG_META_CONTENT_VIEW_CLOSING is sent to the content to notify it of the shutdown. This message is primarily used by **VisContentClass** so it can send MSG_VIS_CLOSE to itself and all its children.

When the view is being destroyed in addition to being closed, the content will receive a MSG_META_CONTENT_VIEW_WIN_CLOSED. You will only need to handle this message if you have cached the view's window handle. If so, your application should throw out the old window handle to avoid any future drawing to the nonexistent window.

The content will then receive a MSG_META_CONTENT_SET_VIEW again (see above), this time with a null value passed as the view's optr. As with the MSG_META_CONTENT_SET_VIEW received when the view started up, your Process object will not need to intercept this message.

9.4.2 Documents in a View

The coordinate space is the graphics space owned by your geode for drawing. Your actual data may occupy any portion of it, but the view considers the entire space to be valid. However, you can constrain the view to a certain portion of the coordinate space—for example, if your data only fills 8.5 inches by 11 inches, you can instruct the view to scroll only in that 8.5 by 11 area (or any smaller or larger portion of it).

9.4.2.1 Document Coordinates

Applications do all their drawing to a device-independent coordinate space. Document coordinates occur in real-world points (72 points per inch). The default graphics system is 8192 points square, with the origin in the middle. The maximum size of a default document is about 113.8 inches on a side. GEOS uses a "large" graphics space as well, represented by 32-bit coordinates that allow documents up to 940 miles on a side. The GenView's coordinates are all given in these 32-bit values to accommodate either the large or the default system.

Your document's data may fill any or all of the coordinate space. Most small documents will have their upper-left corners at the origin and will occupy only a small portion of a 16-bit coordinate space.

9.4.2.2 Current Origin

GVI_origin, MSG_GEN_VIEW_SET_ORIGIN, MSG_GEN_VIEW_GET_ORIGIN, MSG_GEN_VIEW_SET_ORIGIN_LOW

The GenView must know where in the coordinate space it is located. To do this, it maintains an origin—the upper left coordinates of the window's scrollers (see Figure 9-1). That is, the origin is the upper-left position in the document. The origin is stored in the field *GVI_origin*, a **PointDWFixed** structure containing two **DWFixed** structures. The **PointDWFixed** structure definition is shown below:

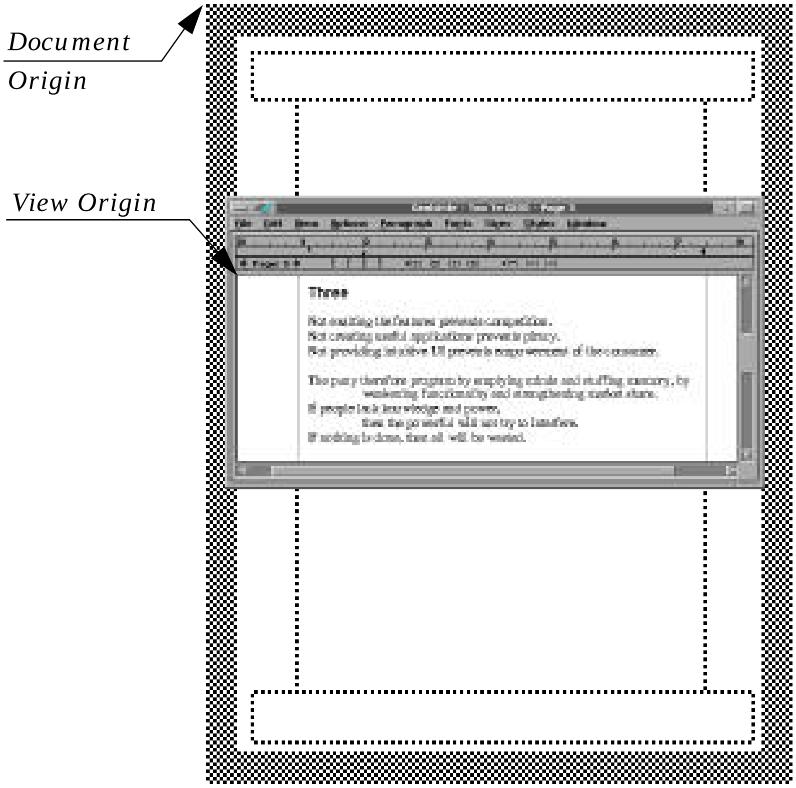
```
typedef struct {
      DWFixed     PDF_x;      /* x origin coord */
      DWFixed     PDF_y;      /* y origin coord */
} PointDWFixed;
```

The view notifies its content each time the origin is changed with the message MSG_META_CONTENT_VIEW_ORIGIN_CHANGED. Typically, the view will also automatically scroll its window to the new origin when the origin changes. You can override this behavior, forcing the view not to scroll on origin changes, by setting the vardata attribute ATTR_GEN_VIEW_DO_NOT_WIN_SCROLL. This attribute, in effect, disconnects the scrollers from the visible position of the document. The notification message MSG_META_CONTENT_VIEW_ORIGIN_CHANGED will still be sent to the content; the window will not scroll, however.

The default origin of the view is (0,0), but this may be set in your Goc code as follows:

```
GVI_origin = {
    xPosition,    /* x coordinate of origin */
    yPosition     /* y coordinate of origin */
}
```

You can also change the origin during execution either by causing the view to scroll or by sending a MSG_GEN_VIEW_SET_ORIGIN. To retrieve the view's current origin, use MSG_GEN_VIEW_GET_ORIGIN.



Three

Not exalting the features prevents competition.
Not creating useful applications prevents piracy.
Not providing intuitive UI prevents empowerment of the consumer.

The party therefore programs by emptying minds and stuffing memory, by
weakening functionality and strengthening market share.
If people lack knowledge and power,
then the powerful will not try to interfere.
If nothing is done, then all will be wasted.

Figure 9-1 *Various Coordinate Spaces*

The view used in GeoWrite (actually a child of a GenDisplay object) displays a portion of the entire document. The upper-left of the document is the document origin, and the upper-left of the view window is the view's origin, stored in the GVI_origin instance field.

■ MSG_GEN_VIEW_SET_ORIGIN

```
void     MSG_GEN_VIEW_SET_ORIGIN(@stack
         sdword yOrigin,          /* y coordinate of new origin */
         sdword xOrigin);         /* x coordinate of new origin */
```

This message changes the view's current origin, causing it to scroll immediately to the new location. The origin represents the upper-left corner of the view in the scrollable document space and is made up of two signed long integers. Each coordinate must be in document coordinates (points).

Source: Unrestricted.

Destination: Any GenView object.

Parameters: *yOrigin* The new Y coordinate of the origin.

xOrigin The new X coordinate of the origin.

Return: Nothing.

Interception: Generally not intercepted.

See Also: MSG_GEN_VIEW_SCROLL, MSG_GEN_VIEW_SCROLL_..., MSG_GEN_VIEW_MAKE_RECT_VISIBLE

■ MSG_GEN_VIEW_GET_ORIGIN

```
void     MSG_GEN_VIEW_GET_ORIGIN(
         PointDWord *origin);
```

This message returns a **PointDWord** structure containing the current X and Y coordinates of the view's origin.

Source: Unrestricted.

Destination: Any GenView object.

Parameters: *origin* A pointer to an empty **PointDWord** structure.

Return: The *origin* pointer will point to a filled structure containing the X and Y coordinates of the current origin. This corresponds to the GenView's *GVI_origin* field.

Interception: Generally not intercepted.

See Also: MSG_GEN_VIEW_GET_DOC_BOUNDS, MSG_GEN_VIEW_GET_DOC_SIZE

■ MSG_GEN_VIEW_SET_ORIGIN_LOW

```
void        MSG_GEN_VIEW_SET_ORIGIN_LOW(@stack
            sdword              yOrigin,
            sdword              xOrigin);
```

This low-level scroll message is just like MSG_GEN_VIEW_SET_ORIGIN except that it will not propagate to linked views.

Source: Unrestricted—usually encapsulated by MSG_GEN_VIEW_SET_ORIGIN for later dispatch with MSG_GEN_VIEW_SEND_TO_HLINK or MSG_GEN_VIEW_SEND_TO_VLINK handler.

Destination: Any GenView.

Parameters: *yOrigin* The new vertical origin of the GenView, in document coordinates. For no change, pass GVSOL_NO_CHANGE.

xOrigin The new horizontal origin of the GenView, in document coordinates. For no change, pass GVSOL_NO_CHANGE.

Return: Nothing.

Interception: Generally not intercepted.

Structures: GVSOL_NO_CHANGE is equal to 0x8000000.

9.4.2.3 Document Bounds

GVI_docBounds, MSG_GEN_VIEW_SET_DOC_BOUNDS, MSG_GEN_VIEW_GET_DOC_BOUNDS

Because most documents will not actually require the entire document space, the view has an instance data field that defines the view's scrollable boundaries. This field is a **RectDWord** structure containing four signed dwords, each of which represents a specific boundary of the document. The **RectDWord** structure is shown below:

```
typedef struct {
      sdword      RD_left;    /* left bound */
      sdword      RD_top;     /* top bound */
      sdword      RD_right;   /* right bound */
      sdword      RD_bottom;  /* bottom bound */
} RectDword;
```

The document bounds are stored in the view's *GVI_docBounds* instance field. The defaults are shown in the following code; note that each of the values in the field must be in document coordinates.

```
GVI_docBounds = {0, 0, 0, 0}; /* zero size */
```

To change the scrollable document bounds at run-time, send the view a message MSG_GEN_VIEW_SET_DOC_BOUNDS. To retrieve the current document bounds, send MSG_GEN_VIEW_GET_DOC_BOUNDS.

■ MSG_GEN_VIEW_SET_DOC_BOUNDS

```
void    MSG_GEN_VIEW_SET_DOC_BOUNDS(@stack
        sdword bottom,          /* new bottom document bound */
        sdword right,           /* new right document bound */
        sdword top,             /* new top document bound */
        sdword left);           /* new left document bound */
```

This message sets the view's scrollable document bounds to those passed. Each of the four parameters must be given in document coordinates (points), and all are signed long integers. If the new document bounds cause the view to show invalid coordinates, the view will scroll until it is showing valid document space.

Source: Unrestricted.

Destination: Any GenView object.

Parameters: *bottom*, *right*, *top*, *left*
As described in the header above.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_VIEW_GET_DOC_BOUNDS

```
void    MSG_GEN_VIEW_GET_DOC_BOUNDS(
        RectDWord *bounds);
```

This message returns the current boundaries of the view's scrollable document space.

Source: Unrestricted.

Destination: Any GenView object.

Parameters: *bounds* A pointer to an empty **RectDWord** structure, to be filled in by the method.

Return: The structure pointed to by *bounds* will be filled with the current scrollable document boundaries of the GenView, as stored in *GVI_docBounds*.

Interception: Generally not intercepted.

9.4.2.4 Current Coordinates in the View

MSG_GEN_VIEW_GET_VISIBLE_RECT

To learn what visible rectangle the view is currently displaying, send it a MSG_GEN_VIEW_GET_VISIBLE_RECT. This message will return the current visible rectangle in document coordinates (a **RectDWord** structure). If the view is not on the screen, the message will return a structure full of zeros.

■ MSG_GEN_VIEW_GET_VISIBLE_RECT

```
void     MSG_GEN_VIEW_GET_VISIBLE_RECT(
         RectDWord *rect);
```

This message returns the rectangle currently visible in the view, in document coordinates.

Source: Unrestricted.

Destination: Any GenView object.

Parameters: *rect* A pointer to an empty **RectDWord** structure to be filled in by the method.

Return: The structure pointed to by rect will contain the bounds of the visible portion of the document. If the view window is not visible on the screen, the structure will contain all zeros.

Interception: Generally not intercepted.

Tips: If your document is particularly complex, you might want to call this message and draw only the visible portions upon receiving either MSG_META_EXPOSED or MSG_VIS_DRAW.

9.4.3 Drawing the Document

MSG_GEN_VIEW_GET_WINDOW, MSG_GEN_VIEW_REDRAW_CONTENT

Displaying your document inside a view is as easy as drawing the document. Any time part of the view becomes invalid (due to scrolling or window movement, for example), the content will receive a MSG_META_EXPOSED. Your content object must have a handler for MSG_META_EXPOSED. See Code Display 9-2 on page ◆ 509 for an example of handling MSG_META_EXPOSED.

You may also cause your application to redraw the window at any time. However, you will need to either save the window handle when the view is created (see MSG_META_CONTENT_VIEW_WIN_OPENED) or retrieve it with MSG_GEN_VIEW_GET_WINDOW. When your content has the window handle, it can create a GState for that window and draw to it as if a MSG_META_EXPOSED had been sent; drawing the changed portion of the document will change the drawing on the screen. Note that the content should not send itself a MSG_META_EXPOSED—this message should be generated only by the window system because the window system will then know which parts of the screen have become invalid. For a self-generated MSG_META_EXPOSED to work, you must also get the view's visible rectangle (with MSG_GEN_VIEW_GET_VISIBLE_RECT, above) and call **GrInvalRect()** to invalidate the view's window.

A simpler way to force a redraw is to make the GenView redraw its entire document with MSG_GEN_VIEW_REDRAW_CONTENT. One example of when this might be used is a "Refresh" or "Redraw" option; this causes the window to be redrawn in order to clean up anything extra that may be left on the screen. Note that this is an option provided by the GenViewControl object.

■ MSG_GEN_VIEW_GET_WINDOW

```
WindowHandle MSG_GEN_VIEW_GET_WINDOW();
```

This message returns the window handle of the view window. As an alternative, you can store the view's window handle when MSG_META_CONTENT_VIEW_WIN_OPENED is sent upon view startup (see section 9.4.1 on page 525) and discard it when you receive MSG_META_CONTENT_VIEW_CLOSING.

Source: Unrestricted.

Destination: Any GenView object.

Parameters: None.

Return: The window handle of the GenView's window.

Interception: Generally not intercepted.

See Also: MSG_META_CONTENT_VIEW_WIN_OPENED, MSG_META_CONTENT_VIEW_CLOSING

■ MSG_GEN_VIEW_REDRAW_CONTENT

```
void        MSG_GEN_VIEW_REDRAW_CONTENT();
```

This message causes the GenView to redraw its entire content. This is essentially the same as exposing the entire view window.

Source: Unrestricted—typically the GenViewControl

Destination: Any GenView object.

Interception: Generally not intercepted.

9.4.4 Document and View Size

Typically, the document's size will be independent of the view's size. The visual relationship between the two is maintained with the use of scrollers. In these situations, the view will size itself according to the size of its parent Primary window. However, you can change the sizing behavior of the view simply by setting attributes or applying hints.

9.4.4.1 Setting the View's Size

```
HINT_INITIAL_SIZE, HINT_MINIMUM_SIZE, HINT_MAXIMUM_SIZE,
HINT_FIXED_SIZE
```

Although the view normally sizes itself to fit the specifications of its parent window, you can change this behavior by applying one of four hints or by setting vertical or horizontal attributes.

- HINT_INITIAL_SIZE
  This hint sets the initial size of the view to a given width and height. The view window will never shrink smaller than the size set with this hint.
- HINT_MINIMUM_SIZE
  This hint sets the smallest possible size of the view window. The view's

initial size is unaffected by this hint except to ensure that the parent Primary opens large enough to accommodate the view window.

- HINT_MAXIMUM_SIZE
  This hint sets the maximum allowable size of the view window. This may be used in place of or in addition to the document bounds.

- HINT_FIXED_SIZE
  This hint has the effect of setting initial, minimum, and maximum sizes at once to set a permanent size for the view window.

All the above hints deal with the structure **CompSizeHintArgs**. This structure allows specification of size in any of a number of different formats including pixels, percentage of screen size, multiples of character widths, number of text lines, and others.

You may also set GVDA_NO_LARGER_THAN_CONTENT and/or GVDA_NO_SMALLER_THAN_CONTENT in the *GVI_horizAttrs* and *GVI_vertAttrs* attribute fields to ensure that the view stays with the content's size. These attributes can be set as follows:

```
GVI_horizAttrs = @default
                | GVDA_NO_LARGER_THAN_CONTENT
                | GVDA_NO_SMALLER_THAN_CONTENT;
GVI_vertAttrs = @default
                | GVDA_NO_LARGER_THAN_CONTENT
                | GVDA_NO_SMALLER_THAN_CONTENT;
```

The above lines will ensure that the view will remain exactly the same size as the content (within screen boundaries).

You can also instruct the view window to snap its width or height to a multiple of the current horizontal or vertical increment. To do this, set the *GVI_vertAttrs* or *GVI_horizAttrs* attribute GVDA_SIZE_A_MULTIPLE_OF_INCREMENT as below:

```
GVI_horizAttrs = @default
                | GVDA_SIZE_A_MULTIPLE_OF_INCREMENT;
GVI_vertAttrs = @default
                | GVDA_SIZE_A_MULTIPLE_OF_INCREMENT;
```

9.4.4.2 Leaving Room for Scrollers

```
HINT_VIEW_LEAVE_ROOM_FOR_VERT_SCROLLER,
HINT_VIEW_LEAVE_ROOM_FOR_HORIZ_SCROLLER
```

Occasionally, you may want to leave extra room for scroller objects even when your view will not have scrollers. You can do so with the hints HINT_VIEW_LEAVE_ROOM_FOR_VERT_SCROLLER and HINT_VIEW_LEAVE_ROOM_FOR_HORIZ_SCROLLER. Although these hints will not affect the size of the view's window, they will affect the view's geometry with respect to its parent window.

9.4.4.3 Adjusting the view's Size

```
MSG_GEN_VIEW_CALC_WIN_SIZE
```

When determining its size initially, the view sends itself the message MSG_GEN_VIEW_CALC_WIN_SIZE. This message, while not useful to applications in and of itself, may be subclassed to get special window sizing behavior. When this message is sent, it includes a suggested window size; by subclassing the GenView and altering the functionality of this message, you can return a different suggested size. The new suggestion will be used to set the view's new size, except at very small sizes—the view will make itself large enough to show scrollers and other children if required.

■ MSG_GEN_VIEW_CALC_WIN_SIZE

```
SizeAsDWord MSG_GEN_VIEW_CALC_WIN_SIZE(
        word   width,          /* suggested new width of view window */
        word   height);        /* suggested new height of view window */
```

This message is sent by the view to itself when it is first being initialized. It takes a suggested size and returns a new suggested size, adjusted for specific conditions.

Source: Unrestricted.

Destination: Any GenView object.

Parameters: *width* The new suggested width of the view window.

height The new suggested height of the view window.

Return: A dword representing both the height and the width determined by the method. The width occupies the low word, and the height occupies the high word.

Interception: If you require special sizing behavior from your GenView window, subclass this method. Be sure to call the superclass in the handler.

9.4.5 Document Scaling

GVI_scaleFactor, MSG_GEN_VIEW_GET_SCALE_FACTOR,
MSG_GEN_VIEW_SET_SCALE_FACTOR, MSG_GEN_VIEW_SCALE_LOW,
ATTR_GEN_VIEW_PAGE_SIZE,
ATTR_GEN_VIEW_SCALE_TO_FIT_BASED_ON_X,
ATTR_GEN_VIEW_SCALE_TO_FIT_BOTH_DIMENSIONS

Many applications implement scaling to allow users to view and modify their documents at various scale factors. This scaling behavior, apart from the implementation of the view menu, is almost entirely internal to the view. The application simply needs to instruct the view in what scale factor to use; the view will provide the proper geometry and math involved in displaying, scrolling, and resizing your document.

Every view has a scale factor stored as a **PointWWFixed** structure in the *GVI_scaleFactor* instance field. The default scale factor is one, or 100 percent. A scale factor of two means that your document will appear twice as large to the user (200 percent normal size). You can set the scale factor in your Goc code and change it with MSG_GEN_VIEW_SET_SCALE_FACTOR.

The **PointWWFixed** structure contains two fields, each of which is a **WWFixed** structure. Both **WWFixed** and **PointWWFixed** are shown below.

```
typedef struct {
      word       WWF_frac;   /* 16 bits decimal */
      word       WWF_int;    /* 16 bits integral */
} WWFixed;
typedef struct {
      WWFixed    PF_x;       /* x scale factor */
      WWFixed    PF_y;       /* y scale factor */
} PointWWFixed;
```

A scale factor of 2.5 (250%) in both dimensions would be declared as follows:

```
GVI_scaleFactor = {{5, 2}, {5, 2}};
```

A scale factor of 100% in the horizontal and 250% in the vertical would be declared as follows:

```
GVI_scaleFactor = {{0, 1}, {5, 2}};
```

To change the scale factor at run-time (for example, when a user clicks on a Zoom button), send MSG_GEN_VIEW_SET_SCALE_FACTOR to the view, passing the proper scale factors. Additionally, you may choose a point on the screen around which to scale. You may also retrieve the current scale factor of a view with the message MSG_GEN_VIEW_GET_SCALE_FACTOR. Note that the GenViewControl has many interactive scaling and zoom features and tools available.

Another way to ensure a certain point is on the screen after scaling is to force it there by causing the view to scroll after scaling. You can suspend the window update, scale the view, then cause the view to scroll with a MSG_GEN_VIEW_MAKE_RECT_VISIBLE. When you then unsuspend the window update, the view will draw with the proper point in the proper place. See section 9.4.7 on page 543 for an in-depth discussion of scrolling the view.

In addition to the dynamic scaling described above, you can have your view scale itself automatically to fit the entire document in its window. To do so, set the *GVI_attrs* flag GVA_SCALE_TO_FIT. This will cause the view to scale itself so the entire document fits vertically in the view; the horizontal scale factor will change to correspond with the new vertical scale factor. You can alter this behavior with three different vardata attributes:

ATTR_GEN_VIEW_PAGE_SIZE
This attribute takes an argument of type **XYSize** (shown below). The view will scale itself to fit this page size rather than its entire document size.

```
typedef struct {
    word    XYS_width;  /* width of page */
    word    XYS_height; /* height of page */
} XYSize;
```

ATTR_GEN_VIEW_SCALE_TO_FIT_BASED_ON_X
This attribute causes the view to scale itself based on the horizontal size rather than the vertical size. The vertical scale factor will follow the horizontal scale factor.

ATTR_GEN_VIEW_SCALE_TO_FIT_BOTH_DIMENSIONS
This attribute causes the view to scale both dimensions independently when scaling to fit. The horizontal and vertical scale factors may end up different.

■ MSG_GEN_VIEW_SET_SCALE_FACTOR

9.4

```
void    MSG_GEN_VIEW_SET_SCALE_FACTOR(@stack
        sdword yOrigin,             /* New Y origin coordinate after scaling */
        sdword xOrigin,             /* New X origin coordinate after scaling */
        ScaleViewType scaleType,    /* Determines Positioning after scaling */
        WWFixedAsDWord yScaleFactor,     /* New scale factor on Y axis */
        WWFixedAsDWord xScaleFactor);    /* New scale factor on X axis */
```

This message changes the view's scale factor, causing it to redraw at the new scale. The view will continue to display your document at the new scale until it is either shut down or instructed to change the scale factor again.

Source: Unrestricted.

Destination: Any GenView object.

Parameters: *yOrigin* New vertical origin of the view window after scaling, if SVT_AROUND_POINT is passed in *scaleType*. Ignored otherwise.

xOrigin New horizontal origin of the view window after scaling, if SVT_AROUND_POINT is passed in *scaleType*. Ignored otherwise.

scaleType A value of **ScaleViewType** indicating how the view is to react after scaling. The three allowable values are described below, under “Structures.”

yScaleFactor The new vertical scale factor. To create a **WWFixedAsDWord** structure, use the macro **MakeWWFixed()**.

xScaleFactor The new horizontal scale factor. To create a **WWFixedAsDWord** structure, use the macro **MakeWWFixed()**.

Return: Nothing.

Interception: Generally not intercepted. If you want to alter scaling behavior, you may intercept this message to change the parameters. Be sure to call the superclass at the end of the handler.

◆Objects

Structures: The type **ScaleViewType** defines the way the view will be scaled. It is an enumerated type with three possible values:

SVT_AROUND_UPPER_LEFT
Scale the view around the upper-left corner, and keep the origin the same before and after scaling. The *xOrigin* and *yOrigin* parameters will be ignored.

SVT_AROUND_CENTER
Scale the view around its center, keeping the point that is central in the view before scaling in the center after scaling. The *xOrigin* and *yOrigin* parameters will be ignored.

SVT_AROUND_POINT
Scale the view around the document point whose coordinates are provided in *yOrigin* and *xOrigin*, making sure the new coordinates are set as the view's new origin.

■ MSG_GEN_VIEW_GET_SCALE_FACTOR

```
void        MSG_GEN_VIEW_GET_SCALE_FACTOR(
            GetScaleParams *retValue);
```

This message returns the current scale factors of the view.

Source: Unrestricted.

Destination: Any GenView object.

Parameters: *retValue* A pointer to an empty **GetScaleParams** structure to be filled in by the method.

Return: The structure pointed to by *retValue* will be filled with the current scale factor of the view.

Interception: Generally not intercepted.

Structures: The **GetScaleParams** structure is made up of two **WWFixedAsDWord** values. The high word of each represents the fractional (decimal) portion of the scale factor, and the low word represents the integral portion. The **GetScaleParams** structure definition is shown below:

```
typedef struct {
      WWFixedAsDWord    GSP_yScaleFactor; /* y */
      WWFixedAsDWord    GSP_xScaleFactor; /* x */
} GetScaleParams;
```

■ MSG_GEN_VIEW_SCALE_LOW

```
void      MSG_GEN_VIEW_SCALE_LOW(@stack
          sdword             yOrigin,
          sdword             xOrigin,
          ScaleViewType      scaleType,
          WWFixedAsDWord     yScaleFactor,
          WWFixedAsDWord     xScaleFactor);
```

This low-level message is similar to MSG_GEN_VIEW_SET_SCALE_FACTOR except that it does not propagate to linked views. It is generally encapsulated by the GenView when another scaling message is received, then dispatched to linked views later.

Source: Sent by the GenView to itself.

Destination: Any GenView object.

Parameters: See MSG_GEN_VIEW_SET_SET_SCALE_FACTOR.

Return: Nothing.

Interception: Should not be intercepted.

9.4.6 Children of the View

`HINT_SEEK_X_SCROLLER_AREA`, `HINT_SEEK_Y_SCROLLER_AREA`, `HINT_SEEK_LEFT_OF_VIEW`, `HINT_SEEK_RIGHT_OF_VIEW`, `HINT_SEEK_TOP_OF_VIEW`, `HINT_SEEK_BOTTOM_OF_VIEW`

Children of GenView objects always appear at a specified edge of the view; you may decide which children appear on the top, left, right, or bottom, but all children must appear along one of the view's edges. Additionally, you may request that the children be placed next to the view's scrollers; for example, if you wanted a page number to appear at the foot of the view, you might define a GenGlyph that seeks the horizontal scroller.

Each child's placement is determined independently of the others. Each child, therefore, must have a hint that describes where in relation to the view it would like to appear. If the requested configuration is supported by the specific UI, the object will appear where requested. If the specific UI does not support the desired configuration, the child will likely appear below the view. The view may have more than one child. If no hint of location is given for any child, that child will appear with the horizontal scroller.

The location hints (defined by **GenClass**) available to a GenView's children include

- HINT_SEEK_X_SCROLLER_AREA
  The object should be placed with the horizontal scroller object.
- HINT_SEEK_Y_SCROLLER_AREA
  The object should be placed with the vertical scroller object.
- HINT_SEEK_LEFT_OF_VIEW
  The object should be forced to the left side of the view.
- HINT_SEEK_RIGHT_OF_VIEW
  The object should be forced to the right side of the view.
- HINT_SEEK_TOP_OF_VIEW
  The object should be forced to the top edge of the view.
- HINT_SEEK_BOTTOM_OF_VIEW
  The object should be forced to the bottom edge of the view.

You may also specify your own scroller objects anywhere below the view in the generic object tree. This is described in the section on Scrolling, below.

9.4.7 Scrolling

When a document is larger than the view window, the user must have some means of navigation around it. The most common implementation of this is scrolling. In most situations, scrolling will be automatic when you use a view. However, you can modify the view's scrolling behavior.

9.4.7.1 Removing the Scrollbars

```
HINT_VIEW_REMOVE_SCROLLERS_WHEN_NOT_SCROLLABLE,
HINT_VIEW_SHOW_SCROLLERS_WHEN_NOT_SCROLLABLE
```

You can determine when scrollbars will be visible on your GVDA_SCROLLABLE view in several ways. One is to set the GVDA_DONT_DISPLAY_SCROLLBAR attribute, either dynamically or in your .goc file, to force a scrollbar not to be displayed. Another is to allow the user to use the GenViewControl to turn the scrollbars on or off.

A third is to allow the GenView to update itself dynamically, by using the hints HINT_VIEW_REMOVE_SCROLLERS_WHEN_NOT_SCROLLABLE and HINT_VIEW_SHOW_SCROLLERS_WHEN_NOT_SCROLLABLE.

The first causes the view to remove the scroller objects when scrolling is not needed (i.e. when the entire content or document fits inside the view's window). The second causes the view to show scrollers in all cases, as long as the view is GVDA_SCROLLABLE and GVDA_DONT_DISPLAY_SCROLLBAR is not set.

9.4.7.2 Making the View Scrollable

Views, by default, are not scrollable. To make your view scrollable, set the GVDA_SCROLLABLE attribute in both the *GVI_horizAttrs* and *GVI_vertAttrs* records. It is easiest to do this in your Goc code, as follows:

```
GVI_horizAttrs = @default | GVDA_SCROLLABLE;
GVI_vertAttrs = @default | GVDA_SCROLLABLE;
```

You can also set these attributes during execution by sending the message MSG_GEN_VIEW_SET_DIMENSION_ATTRS to the view. See section 9.3.2 on page 516.

9.4.7.3 Normal Scrolling

There are three basic ways scrolling can be initiated: First, the user can click on a special scroller object. Second, the user can initiate "drag scrolling" by clicking within the view and dragging outside the window's bounds. Third, the view's content object can request a scrolling action. Only the first of these three is activated by default.

When a scrollable view is first created, scroller objects are also instantiated. This is automatic and for the most part transparent to the geode. The scroller objects may be customized or placed in a certain spot in the generic tree: to learn how to do this, see section 9.4.7.7 on page 559.

Scrollers are generic by nature so they can conform to the specific UI in use; in OSF/Motif, for example, scrollers are manifested as scrollbars that position themselves at the right and bottom sides of the view. However, other specific UIs might implement dials, sliders, or some other gadgets for scrolling.

In normal scrolling, the user provides input to a scroller (for example, clicking on the up-arrow of a vertical scrollbar in OSF/Motif). The scroller then sends a message to the view indicating that a scroll should take place. The view calculates the new portion of the document and sends a MSG_META_EXPOSED to its content to indicate that the document must be redrawn. The content then draws its document to the view's window handle, and the view takes care of clipping the document and translating it to the screen.

To take advantage of the normal scrolling features, therefore, you only have to make the view scrollable and respond to MSG_META_EXPOSED.

9.4.7.4 Drag Scrolling

```
MSG_GEN_VIEW_INITIATE_DRAG_SCROLL,
MSG_GEN_VIEW_SET_DRAG_BOUNDS,
HINT_VIEW_IMMEDIATE_DRAG_UPDATES,
HINT_VIEW_DELAYED_DRAG_UPDATES
```

In many cases, applications will want to supplement normal scrolling with drag scrolling, in which a user clicks within the view and drags beyond its edge, causing the view to scroll in that direction. Also, some applications may wish to implement scrolling without the use of scrollers—drag scrolling provides a good way of doing this.

Two hints affect how drag scrolling is implemented. HINT_VIEW_IMMEDIATE_DRAG_UPDATES causes the view to be updated constantly during drags. HINT_VIEW_DELAYED_DRAG_UPDATES causes the view to be updated only at the end of drags. The default behavior is to update periodically during the drag in order to provide visual feedback to the user and to avoid excessive overhead in drawing repeatedly.

To implement drag scrolling, your view must be scrollable. It also must have the GVA_DRAG_SCROLLING attribute of the *GVI_attrs* record set. You can do this in Goc as follows:

```
GVI_horizAttrs = @default | GVDA_SCROLLABLE;
GVI_vertAttrs = @default | GVDA_SCROLLABLE;
GVI_attrs = @default | GVA_DRAG_SCROLLING;
```

You can also set the drag scrolling attribute by sending the view the message MSG_GEN_VIEW_SET_ATTRS (see section 9.3.1 on page 512).

Normally, drag scrolling works only for the select button (in OSF/Motif, the left mouse button). MSG_GEN_VIEW_INITIATE_DRAG_SCROLL allows you to change this. For example, if you received a MSG_META_START_MOVE_COPY (indicating that the user used the direct-action button), you might want to initiate drag scrolling if your application does not support quick-transfer.

Additionally, if you wanted drag scrolling to work only in a portion of your document (for example, if you wanted the user to be able to move an object on the screen but only move it so far), you could set the temporary drag bounds directly after initiating the drag scrolling. This is done by sending the view a MSG_GEN_VIEW_SET_DRAG_BOUNDS.

■ MSG_GEN_VIEW_INITIATE_DRAG_SCROLL

```
void    MSG_GEN_VIEW_INITIATE_DRAG_SCROLL();
```

This message instructs the view to begin drag scrolling until the pressed mouse button is let go by the user. When drag scrolling is enabled, the select button initiates it; this message allows initiation by any mouse button.

Source: Unrestricted—called by an application on a MSG_META_START_… mouse event.

Destination: The GenView in which the mouse button was clicked.

Interception: Generally not intercepted.

■ MSG_GEN_VIEW_SET_DRAG_BOUNDS

```
void    MSG_GEN_VIEW_SET_DRAG_BOUNDS(@stack
        sdword bottom,      /* bottom scrolling bound */
        sdword right,       /* right scrolling bound */
        sdword top,         /* top scrolling bound */
        sdword left);       /* left scrolling bound */
```

This message sets a temporary rectangle in which drag scrolling can operate if the user should not drag scroll across the entire document.

Source: Unrestricted—called by an application on the initiation of drag scrolling.

Destination: The GenView in which the drag scrolling is underway.

Parameters: *bottom, right, top, left*
The bounds (in document coordinates) of the

rectangle limiting scrolling. These temporary bounds will be used only during the current drag scroll. The restriction will then be lifted.

Return: Nothing.

Interception: Generally not intercepted.

9.4.7.5 Modifying Scrolling

Many applications may find a need to scroll the view without user input or to modify the scrolling behavior of the view. For example, if your application has a trigger or menu item that brings a certain portion of the document onto the screen, you will need to instruct the view to scroll to that point, or perhaps your application must scroll by different increments depending on the situation.

Scrolling from the Content

MSG_GEN_VIEW_SCROLL_...

All typical messages sent by scrollers to the view are also available for sending by applications and other geodes. These messages are listed below. Note: With the exception of MSG_GEN_VIEW_SCROLL, these messages work only when the view is visibly initialized. This means that a view that is not on the screen will ignore these messages.

■ MSG_GEN_VIEW_SCROLL

```
void      MSG_GEN_VIEW_SCROLL(@stack
          sdword yOffset,        /* amount to scroll vertically */
          sdword xOffset);       /* amount to scroll horizontally */
```

This message causes the view to scroll a given amount in both the X and Y directions.

Source: Unrestricted.

Destination: Any GenView object.

Parameters: *yOffset* The signed vertical amount to scroll.

xOffset The signed horizontal amount to scroll.

Return: Nothing.

Interception: Generally not intercepted. If you want to alter scrolling behavior, see "Tracking the Scrolling" on page 554.

■ MSG_GEN_VIEW_SCROLL_TOP

```
void        MSG_GEN_VIEW_SCROLL_TOP();
```

Causes the view to scroll to the top of the document.

Source: Unrestricted.

Destination: Any GenView object—views not visible will ignore these messages.

Interception: Generally not intercepted.

■ MSG_GEN_VIEW_SCROLL_PAGE_UP

```
void        MSG_GEN_VIEW_SCROLL_PAGE_UP();
```

Causes the view to scroll up one window height.

Source: Unrestricted.

Destination: Any GenView object—views not visible will ignore these messages.

Interception: Generally not intercepted.

■ MSG_GEN_VIEW_SCROLL_UP

```
void        MSG_GEN_VIEW_SCROLL_UP();
```

Causes the view to scroll up one increment.

Source: Unrestricted.

Destination: Any GenView object—views not visible will ignore these messages.

Interception: Generally not intercepted.

■ MSG_GEN_VIEW_SCROLL_SET_Y_ORIGIN

```
void        MSG_GEN_VIEW_SCROLL_SET_Y_ORIGIN(
            sdword yOrigin);
```

Causes the view to scroll to a given vertical point and sets the Y component of the origin to the passed value. It keeps the X component of the origin fixed.

Source: Unrestricted.

Destination: Any GenView object—views not visible will ignore these messages.

Interception: Generally not intercepted.

■ MSG_GEN_VIEW_SCROLL_DOWN

```
void        MSG_GEN_VIEW_SCROLL_DOWN();
```

Causes the view to scroll down one increment.

Source: Unrestricted.

Destination: Any GenView object—views not visible will ignore these messages.

Interception: Generally not intercepted.

■ MSG_GEN_VIEW_SCROLL_PAGE_DOWN

```
void        MSG_GEN_VIEW_SCROLL_PAGE_DOWN();
```

Causes the view to scroll down one window height.

Source: Unrestricted.

Destination: Any GenView object—views not visible will ignore these messages.

Interception: Generally not intercepted.

■ MSG_GEN_VIEW_SCROLL_BOTTOM

```
void        MSG_GEN_VIEW_SCROLL_BOTTOM();
```

Causes the view to scroll to the bottom of the document.

Source: Unrestricted.

Destination: Any GenView object—views not visible will ignore these messages.

Interception: Generally not intercepted.

■ MSG_GEN_VIEW_SCROLL_LEFT_EDGE

```
void        MSG_GEN_VIEW_SCROLL_LEFT_EDGE();
```

Causes the view to scroll to the document’s left edge.

Source: Unrestricted.

Destination: Any GenView object—views not visible will ignore these messages.

Interception: Generally not intercepted.

■ MSG_GEN_VIEW_SCROLL_PAGE_LEFT

```
void        MSG_GEN_VIEW_SCROLL_PAGE_LEFT();
```

Causes the view to scroll left one window width.

Source: Unrestricted.

Destination: Any GenView object—views not visible will ignore these messages.

Interception: Generally not intercepted.

■ MSG_GEN_VIEW_SCROLL_LEFT

```
void        MSG_GEN_VIEW_SCROLL_LEFT();
```

Causes the view to scroll left one increment.

Source: Unrestricted.

Destination: Any GenView object—views not visible will ignore these messages.

Interception: Generally not intercepted.

■ MSG_GEN_VIEW_SCROLL_SET_X_ORIGIN

```
void        MSG_GEN_VIEW_SCROLL_SET_X_ORIGIN(
            sdword xOrigin);
```

Causes the view to scroll to a given horizontal point and sets the X component of the origin to the passed value. It keeps the Y component of the origin fixed.

Source: Unrestricted.

Destination: Any GenView object—views not visible will ignore these messages.

Interception: Generally not intercepted.

■ MSG_GEN_VIEW_SCROLL_RIGHT

```
void        MSG_GEN_VIEW_SCROLL_RIGHT();
```

Causes the view to scroll right one increment.

Source: Unrestricted.

Destination: Any GenView object—views not visible will ignore these messages.

Interception: Generally not intercepted.

■ MSG_GEN_VIEW_SCROLL_PAGE_RIGHT

```
void        MSG_GEN_VIEW_SCROLL_PAGE_RIGHT();
```

Causes the view to scroll right one window width.

Source: Unrestricted.

Destination: Any GenView object—views not visible will ignore these messages.

Interception: Generally not intercepted.

■ MSG_GEN_VIEW_SCROLL_RIGHT_EDGE

```
void	MSG_GEN_VIEW_SCROLL_RIGHT_EDGE();
```

Causes the view to scroll to the document's right edge.

Source: Unrestricted.

Destination: Any GenView object—views not visible will ignore these messages.

Interception: Generally not intercepted.

Making a Rectangle Visible

MSG_GEN_VIEW_MAKE_RECT_VISIBLE

You can easily make any portion of your document visible with the passage of a single message. For example, if you were displaying a map of the United States and the user clicked on a menu item to show the area around Cincinnati, you could easily determine the coordinates of Cincinnati and instruct the view to scroll there.

To make a certain rectangle of the document visible in the view, pass the message MSG_GEN_VIEW_MAKE_RECT_VISIBLE to the view. You can include a percentage to indicate how far through the rectangle the view should scroll—zero percent causing the rectangle to appear just at the near edge of the view and 100 percent causing the rectangle to appear just at the far edge. Fifty percent causes the object to be centered in the view window.

Normally, this message will not cause any scrolling if the rectangle is already on the screen. However, you can indicate that this message should always cause scrolling (useful if you want to center a rectangle that is already partially on the screen).

■ MSG_GEN_VIEW_MAKE_RECT_VISIBLE

```
void    MSG_GEN_VIEW_MAKE_RECT_VISIBLE(@stack
        word   yFlags,       /* MakeRectVisibleFlags for y dimension */
        word   yMargin,      /* percentage to scroll onto screen in y */
        word   xFlags,       /* MakeRectVisibleFlags for x dimension */
        word   xMargin,      /* percentage to scroll onto screen in x */
        sdword bottom,       /* bottom bound of the target rectangle */
        sdword right,        /* right bound of the target rectangle */
        sdword top,          /* top bound of the target rectangle */
        sdword left);        /* left bound of the target rectangle */
```

This message causes the view to scroll until a given portion of a passed rectangle is visible in the display window. If the rectangle is already partially visible, the view will not scroll unless instructed to do so by the flags arguments. If the view is not visibly initialized, this message will have no effect.

Source: Unrestricted.

Destination: Any GenView object.

Parameters: *yFlags* A record of flags indicating when and how scrolling occurs. See below.

yMargin The amount of the rectangle to be scrolled onto the screen in the vertical dimension. This is a percentage of the screen height and is based from the edge of the screen closest to the rectangle. For example, a *yMargin* of 50% centers the rectangle vertically; a *yMargin* of 100% puts the rectangle at the far edge of the screen from where it started. The percentages available are shown below.

xFlags A record of flags indicating when and how scrolling occurs. See below.

xMargin The amount of the rectangle to be scrolled onto the screen in the horizontal dimension. This is a percentage of the screen width and is based from the edge of the screen closest to the rectangle. For example, an *xMargin* of 50% centers the rectangle horizontally; an *xMargin* of 100% puts the rectangle at the far edge of the screen from where it started. The percentages available are shown below.

bottom, *right*, *top*, *left*
The bounds of the rectangle to be made visible. These should be in document coordinates.

Return: Nothing.

Structures: The *yFlags* and *xFlags* parameters are of type **MakeRectVisibleFlags**; this type has the following values:

MRVF_ALWAYS_SCROLL
This flag indicates that the view should scroll even when the rectangle is already partially visible.

MRVF_USE_MARGIN_FROM_TOP_LEFT
This flag indicates that margin percentages should be taken relative to the top and left edges of the view rather than the edges closest to the defined rectangle.

The percentages passed in *yMargin* and *xMargin* have a special format. You can use a predefined constant, or you can pass custom percentage. To use your own percentage, multiply the decimal representing the desired percentage (e.g. ".50" represents 50 percent) by the hexadecimal number 0xffff; or, instead of doing that math, you can use one of the following system-provided constants:

```
MRVM_0_PERCENT
MRVM_25_PERCENT
MRVM_50_PERCENT
MRVM_75_PERCENT
MRVM_100_PERCENT
```

Interception: Generally not intercepted.

Handling Complex Scrolling Operations

MSG_GEN_VIEW_SUSPEND_UPDATE, MSG_GEN_VIEW_UNSUSPEND_UPDATE

If you cause the view to go through several scrolling operations in rapid order, you may want to suspend the view from sending a MSG_META_EXPOSED after each—it is much better to wait until all the updates have been made before the redrawing occurs.

To suspend a view's updates temporarily, send it the message MSG_GEN_VIEW_SUSPEND_UPDATE. After you have finished sending the scrolling messages, unsuspend the view's updates by sending MSG_GEN_VIEW_UNSUSPEND_UPDATE.

■ MSG_GEN_VIEW_SUSPEND_UPDATE

```
void        MSG_GEN_VIEW_SUSPEND_UPDATE();
```

This message causes the view to suspend sending MSG_META_EXPOSED until it receives MSG_GEN_VIEW_UNSUSPEND_UPDATE. Suspend–unsuspend pairs can be nested.

Source: Unrestricted.

Destination: Any GenView object.

Interception: Generally not intercepted.

■ MSG_GEN_VIEW_UNSUSPEND_UPDATE

```
void        MSG_GEN_VIEW_UNSUSPEND_UPDATE();
```

This message allows the view to send a MSG_META_EXPOSED again after being suspended with MSG_GEN_VIEW_SUSPEND_UPDATE.

Source: Unrestricted.

Destination: Any GenView object.

Interception: Generally not intercepted.

Warnings: It is an error to send this message to a non-suspended view.

9.4.7.6 Tracking the Scrolling

```
MSG_META_CONTENT_TRACK_SCROLLING,
MSG_GEN_VIEW_SETUP_TRACKING_ARGS,
MSG_GEN_VIEW_TRACKING_COMPLETE
```

For flexibility, the view allows applications to track scrolling as it happens and alter it before it gets implemented. When GVA_TRACK_SCROLLING is set in *GVI_attrs*, all scrolling events will be sent to the content object before being implemented. Because there are often many scrolling events, you should avoid track scrolling unless truly necessary.

If you are going to track the scrolling, your content object *must* handle the message MSG_META_CONTENT_TRACK_SCROLLING. This message passes a **TrackScrollingParams** structure containing all the information needed about the upcoming scrolling event. When you receive it, you must do the following three steps:

1 Send MSG_GEN_VIEW_SETUP_TRACKING_ARGS to the view
This message fills in all the information of the **TrackScrollingParams** structure. You must call this routine to ensure that there are no synchronization problems with the scrolling event.

2 Alter the scrolling event
If you choose to alter the scrolling event, change the **TrackScrollingParams** structure to the appropriate values.

3 Send MSG_GEN_VIEW_TRACKING_COMPLETE to the view
This message updates the **TrackScrollingParams** structure based on your changes, then sends a message to the view with the new scrolling information. The view will then scroll according to your changes.

The **TrackScrollingParams** structure contains complete information about the scrolling event including context information, the view's width and height, the view's document origin, and the action in progress. It also contains a field called *TSP_change*, which is a structure of type **PointDWord** and should, after you change the scrolling, contain an offset from the proposed origin to the new origin you desire. See Code Display 9-4 for the definition of the structure.

Code Display 9-4 TrackScrollingParams and Associated Structures

```
/*
 * The TrackScrollingParams structure contains all the necessary information about
 * a given scrolling event. It is made up of several substructures, each of which
 * is shown below.
 */

/*
 * ScrollAction is an enumerated type, each enumeration of which designates a
 * different type of event. The handler should not change these.
 */
typedef ByteEnum ScrollAction;   /* byte-length enumeration */
/*  SA_NOTHING          * No scrolling action
    SA_TO_BEGINNING     * Scrolling to beginning of document
    SA_PAGE_BACK        * Scrolling back one screen height or width
    SA_INC_BACK         * Scrolling back one increment
    SA_INC_FWD          * Scrolling forward one increment
    SA_DRAGGING         * Drag scrolling is underway
```

```
        SA_PAGE_FWD         * Scrolling forward one page
        SA_TO_END           * Scrolling to end of document
        SA_SCROLL           * Generic scrolling message sent
        SA_SCROLL_INTO      * Scrolling while keeping a point on screen
        SA_INITIAL_POS      * Indicating the initial scrolling position.
                            * Subsequent scroll messages will be relative to
                            * this origin.
        SA_SCALE            * Scaling may cause scrolling
        SA_PAN              * Pan-scrolling is underway. Otherwise
                            * identical to SA_SCROLL
        SA_DRAG_SCROLL      * Drag-scrolling, otherwise like SA_SCROLL
        SA_SCROLL_FOR_SIZE_CHANGE    * Scrolling because view size changed
    */

    /*
     * ScrollFlags is a byte record of flags used to determine the type of scrolling
     * taking place and the context of the scroll.
     */

    typedef ByteFlags ScrollFlags;
    #define SF_VERTICAL     0x80    /* Scrolling is vertical if set. If clear,
                                     * scrolling is horizontal. Invalid for
                                     * ScrollAction types SA_SCROLL_INTO,
                                     * SA_SCROLL, and SA_INITIAL_POS. */
    #define SF_ABSOLUTE     0x40    /* Scrolling is to an absolute point. Set for
                                     * ScrollAction types SA_TO_BEGINNING,
                                     * SA_TO_END, SA_INITIAL_POS, SA_SCROLL_INTO,
                                     * SA_DRAGGING, and some SA_SCROLL. */
    #define SF_DOC_SIZE_CHANGE 0x20 /* Scroll resulted from document size change. */
    #define SF_WINDOW_NOT_SUSPENDED 0x10
                                    /* Flag used internally only. */
    #define SF_SCALE_TO_FIT 0x08    /* Flag used when the View is in scale-to-fit
                                     * mode (often changes scrolling behavior). */
    #define SF_SETUP_HAPPENED  0x04 /* Flag for error checking only. */

    /*
     * The TrackScrollingParams structure contains several elements, each of
     * which is described in the comments below. All the fields will be filled by
     * MSG_GEN_VIEW_SETUP_TRACKING_ARGS; however, only certain fields will be filled
     * by MSG_META_CONTENT_TRACK_SCROLLING. Therefore, you should always send
     * MSG_GEN_VIEW_SETUP_TRACKING_ARGS in the handler for this message.
     */

    typedef struct {
        ScrollAction TSP_action;    /* The type of scrolling underway */
        ScrollFlags  TSP_flags;     /* flags shown above */
        optr         TSP_caller;    /* The sender of the scroll message */
```

9.4

```
    PointDWord  TSP_change;     /* The relative amount being scrolled */
    PointDWord  TSP_newOrigin;  /* The new absolute origin */
    PointDWord  TSP_oldOrigin;  /* The original origin */
    sword       TSP_viewWidth;  /* Current view width */
    sword       TSP_viewHeight; /* Current view height */
} TrackScrollingParams;
```

To set *TSP_change*, you must first understand how the entire **TrackScrollingParams** structure is used. When a scrolling event is being tracked, three sets of information are needed:

◆ The current origin and size of the view. This is represented by the field *TSP_oldOrigin*.

◆ The scrolling event's proposed change to the view's origin. This is represented by the field *TSP_newOrigin*.

◆ The final change to the view's origin after your adjustments have been added. This is represented by a combination of *TSP_oldOrigin* and *TSP_change*. You should set *TSP_change* understanding that it gets added to *TSP_oldOrigin* to produce the final origin of the view.

Tracking the scrolling, however, can introduce noticeable lag in scrolling speed. If you do not need to track the scrolling, you probably shouldn't. However, if you need to track the scrolling for simple or few changes, you should consider subclassing **GenViewClass** and altering the functionality of the method for MSG_META_CONTENT_TRACK_SCROLLING. The GenView's handler for this message simply passes the message on to the content.

If you just need to track changes to the origin, subclass **GenViewClass** and intercept MSG_META_CONTENT_VIEW_ORIGIN_CHANGED.

Code Display 9-5 Handling Track Scrolling

```
/* This message is sent by the view to its content object when track scrolling is
 * enabled and a scrolling event is begun. If you plan to track the scrolling, you
 * MUST handle this message. The first thing your handler should do is send
 * MSG_GEN_VIEW_SETUP_TRACKING_ARGS to the view. This message fills in the
 * TrackScrollingParams structure. The last thing the handler must do is send
 * MSG_GEN_VIEW_TRACKING_COMPLETE to the view. This message locks your changes
 * in and directs the view to implement them.
 * The format of this message is
 *      void (TrackScrollingParams *args);                                */

@message        MyProcessClass, MSG_META_CONTENT_TRACK_SCROLLING {
    @call (args->TSP_caller)::MSG_GEN_VIEW_SETUP_TRACKING_ARGS(args);
        /* Here you can do whatever you want to args->TSP_change. */
    @call MyView::MSG_GEN_VIEW_TRACKING_COMPLETE(&args);
}
```

■ MSG_GEN_VIEW_SETUP_TRACKING_ARGS

```
void    MSG_GEN_VIEW_SETUP_TRACKING_ARGS(
        TrackScrollingParams *args);
```

This message takes the **TrackScrollingParams** structure and fills in remaining fields not passed by MSG_META_CONTENT_TRACK_SCROLLING.

Source: The content object tracking scrolling (in the handler for MSG_META_CONTENT_TRACK_SCROLLING).

Destination: The calling object, found in *args->TSP_caller*.

Parameters: *args* A pointer to the mostly complete structure of **TrackScrollingParams**, to be filled in by the method.

Return: The structure pointed to by *args* will be filled.

Structures: See Code Display 9-4 on page ◆ 555.

Interception: Generally not intercepted.

■ MSG_GEN_VIEW_TRACKING_COMPLETE

```
void    MSG_GEN_VIEW_TRACKING_COMPLETE(
        TrackScrollingParams *args);
```

This message sets the altered scrolling arguments and makes them official, instructing the view to implement them.

Source: The content object tracking scrolling (in its handler for MSG_META_CONTENT_TRACK_SCROLLING).

Destination: The calling object, found in *args->TSP_caller*.

Parameters: *args* A pointer to the completed structure of **TrackScrollingParams**.

Return: Nothing.

Structures: See Code Display 9-4 on page ◆ 555.

Interception: Generally not intercepted.

■ MSG_META_CONTENT_TRACK_SCROLLING

```
void	MSG_META_CONTENT_TRACK_SCROLLING(
	TrackScrollingParams *args);
```

This message is sent by the GenView to its content if the content has requested track-scrolling (with GVA_TRACK_SCROLLING set in the GenView's instance data).

Source: Sent by a GenView object to its content when a scrolling event occurs and the view has GVA_TRACK_SCROLLING set.

Destination: The content of the GenView.

Parameters: *args* A pointer to a **TrackScrollingParams** structure.

Return: Nothing.

Structures: See "TrackScrollingParams and Associated Structures" on page 555.

Interception: Any content that tracks scrolling *must* handle this message. It should send MSG_GEN_VIEW_SETUP_TRACKING_ARGS and MSG_GEN_VIEW_TRACKING_COMPLETE to the object specified in *TSP_caller*.

9.4.7.7 Customizing the Scrollers

```
HINT_VALUE_X_SCROLLER, HINT_VALUE_Y_SCROLLER
```

Although a view's scrollers should remain generic to maintain UI consistency, you may set your own GenValue objects to be the scrollers of your view. This allows special placement of the scrollers not only around the view but also within the generic object tree.

Before creating its own scrollers, the view will search for any objects below it in the generic object tree that have the hint HINT_VALUE_X_SROLLER or HINT_VALUE_Y_SCROLLER. For example, if you wanted to put your vertical scroller on the left of the view rather than on the right, you could add the following code:

```
@object GenViewClass MyView = {
    /* other view attributes */
    GI_comp = MyViewScroller;
}
@object GenRangeClass MyViewScroller = {
    HINT_Y_SCROLLER;
    HINT_SEEK_LEFT_OF_VIEW;
}
```

Custom scrollers do not need to be direct children of the view; instead, they may be any number of generations below the view in the generic object tree.

9.4.8 Monitoring Input

Often, applications will want to receive input—keyboard, mouse, or pen events—through the view. While keyboard events are fairly straightforward, receiving mouse and pen events can be somewhat more complicated. Depending on the types of events you choose to receive, however, input handling can be quite simple.

9.4.8.1 Mouse Events

Many applications will want to receive mouse events through the view. By default, the view's content will receive all pointer events, whether or not a mouse button has been pressed. However, you can determine which types of events you want to receive by setting the appropriate attributes in the *GVI_attrs* field:

- GVA_DONT_SEND_PTR_EVENTS
  This overrides the default and ensures that no mouse (pointer) events are passed on to the content. This is used for optimization when your content will not interact with the user directly.

- GVA_WINDOW_COORDINATE_MOUSE_EVENTS
  This changes the format of the mouse event as received by the content object and is used with large documents. See below for more information about handling input to large documents.

Both of the above attributes may be set either in your geode's Goc code or with MSG_GEN_VIEW_SET_ATTRS. To retrieve which attributes are set, use MSG_GEN_VIEW_GET_ATTRS. Due to overhead incurred by passing input events, you should only set the attributes you need. (See section 9.3.1 on page 512 for more detail on these messages.)

Handling 32-Bit Input Events

Because normal pointer events use the standard GEOS 16-bit coordinate system, large documents must be able to translate these 16-bit events into 32-bit coordinates. The GenView makes this easy for you.

First, set the attribute GVA_WINDOW_COORDINATE_MOUSE_EVENTS in the *GVI_attrs* record. This will change the format of pointer data passed from the view to the content object. The new format will contain offset coordinates describing the pointer's position relative to the view's screen origin (in screen coordinates).

When you receive one of these events (called "large" mouse events), you can apply the simple formula shown in Equation 9-1 to get 32-bit document coordinates. Once you have the coordinates of the pointer event, you can deal with it appropriately.

To keep track of the view's origin and scale factor, you should intercept the messages MSG_META_CONTENT_VIEW_ORIGIN_CHANGED and MSG_META_CONTENT_VIEW_SCALE_FACTOR_CHANGED. You should save the values passed by these messages for use in this equation. (For full details on these messages, see section 25.2.2 on page 1443 of the Objects books.)

$$\frac{ViewWindowCoordinate}{ScaleFactor} + ViewDocumentOrigin$$

Equation 9-1 *Translating Input Events*
ViewWindowCoordinate is the distance, in screen pixels, between the pointer and the view's upper left corner. ScaleFactor is the view's scale factor. ViewDocumentOrigin is the 32-bit origin, in document coordinates. The result of this equation provides 32-bit document coordinates for 16-bit mouse and pointer events. To do the division, use the routine ***GrSDivWWFixed()****.*

Losing Mouse Grabs

During certain operations—for example, quick-transfers—your geode will have the mouse grab. When it loses the grab, it will be notified with a MSG_META_CONTENT_LOST_GADGET_EXCLUSIVE.

Supporting Quick-Transfer

```
MSG_GEN_VIEW_ALLOW_GLOBAL_TRANSFER
```

When using a view and supporting the quick-transfer mechanism, the view will grab the mouse when a quick-transfer operation is invoked. Conclusion of the quick-transfer operation, requires, however, that other objects know when the mouse crosses their bounds; otherwise, no other object in the system could ever become a destination for the transfer.

To relinquish the mouse grab while continuing the transfer operation, you must send MSG_GEN_VIEW_ALLOW_GLOBAL_TRANSFER to the view. The view will then let other objects get pointer events, thus letting them know when they become a potential destination for the transfer.

■ MSG_GEN_VIEW_ALLOW_GLOBAL_TRANSFER

```
void      MSG_GEN_VIEW_ALLOW_GLOBAL_TRANSFER();
```

This message indicates to the view that it should relinquish the mouse grab to allow a quick-transfer operation to continue. Pointer events will then be sent to other objects in the system, allowing them to accept or reject the transfer.

Source: The object with the mouse grab in the view.

Destination: The GenView displaying the caller object.

Interception: Generally not intercepted.

9.4.8.2 Keyboard Events

By default, all keyboard events not handled by the Specific UI library are passed through to the content when the view has the keyboard focus. Only views with the GVA_FOCUSABLE flags set in *GVI_attrs* will ever gain the focus (this is the default). Many character strokes are used for keyboard navigation or as mnemonics or accelerators and will therefore be intercepted by the Specific UI. Those that aren't will be passed on to your content.

You can set up your content to receive all keyboard events before the Specific UI has a chance to intercept them. This can have unpredictable results, however, and should be avoided unless absolutely necessary. To set up your content in this manner, set the flag GVA_SEND_ALL_KBD_CHARS in the view's *GVI_attrs* field. If you handle keyboard events before passing them on, you must be sure to pass every unused MSG_META_KBD_CHAR on to the UI in the form of MSG_META_VUP_KBD_CHAR. If you do not, undesirable synchronization problems could result.

You can indicate that you do not want to receive keyboard releases by setting the attribute GVA_DONT_SEND_KBD_RELEASES in the *GVI_attrs* record. This is provided as an optimization to improve keyboard response in applications. Most applications need to know when a key has been pressed but do not care when it has been released.

A better way to handle accelerator characters that don't appear in your UI objects is to subclass **GenViewClass** and intercept MSG_META_KBD_CHAR; in the handler, take the characters you want and then call the superclass with **@callsuper()** to make sure unused events get passed on. As an alternative, you can subclass **GenApplicationClass** and change the handler for MSG_META_VUP_KBD_CHAR; in the method, take the events you want and then call the superclass with **@callsuper()**.

9.4.8.3 Pen Input and Ink

```
GVI_inkType, MSG_GEN_VIEW_SET_INK_TYPE,
MSG_GEN_VIEW_SET_EXTENDED_INK_TYPE,
MSG_GEN_VIEW_RESET_EXTENDED_INK_TYPE,
ATTR_GEN_VIEW_DOES_NOT_ACCEPT_TEXT_INPUT
```

Pen input will be sent to applications through the Ink library—it is suggested that you read "Pen Object Library," Chapter 21, as well as "Input," Chapter 11 of the Concepts Book, before working with Ink in the view. The GenView accepts Ink input events and can pass them to the content either as ink or as mouse events. How the view determines which type of events are desired is mandated by the value of the instance field *GVI_inkType*.

In addition, some views that get the focus do not accept text input. By default on many pen systems, focusable GenViews will cause an on-screen keyboard to come up on the assumption that the user may wish to enter text to objects within the view. ATTR_GEN_VIEW_DOES_NOT_ACCEPT_TEXT_INPUT allows you to specify that your view does not accept text input and therefore should not cause a floating keyboard to come up. Non-textual keyboard events (e.g. up-arrow used for scrolling) will still be transmitted to the GenView.

The *GVI_inkType* field will contain one of the following values (each is an enumeration of **GenViewInkType**):

GVIT_PRESSES_ARE_NOT_INK
Mouse press events are not Ink input, and Ink input is not expected by the content. This is the default for views.

GVIT_INK_WITH_STANDARD_OVERRIDE
Mouse press events are considered and expected to be Ink events, but they can be overridden by the user in the normal way.

GVIT_PRESSES_ARE_INK
Mouse press events are considered and expected to be Ink.

GVIT_QUERY_OUTPUT

This value indicates that the content will expect Ink events only in certain contexts. Note that if this attribute is set, the content *must* handle the MSG_META_QUERY_IF_PRESS_IS_INK message.

The *GVI_inkType* field may be set either in your Goc code or with the message MSG_GEN_VIEW_SET_INK_TYPE, shown below.

Applications that handle Ink input can also set extended information about the Ink. The type of extended information you can set includes the color of the Ink as it's drawn, the thickness of the Ink brush, etc. Use the messages MSG_GEN_VIEW_SET_EXTENDED_INK_TYPE and MSG_GEN_VIEW_RESET_EXTENDED_INK_TYPE, shown below.

You can also pre-set the extended Ink information with the vardata attribute ATTR_GEN_VIEW_INK_DESTINATION_INFO. This attribute has an extra data structure of type **InkDestinationInfoParams**, shown below.

```
typedef struct {
      optr        IDIP_dest;
                        /* destination object
                         * for Ink output. */
      word        IDIP_brushSize;
                        /* Size of Ink brush.
                         * High byte is x size,
                         * Low byte is y size. */
      byte        IDIP_color; /* Color of Ink */
      Boolean     IDIP_createGState;
                              /* Boolean indicating
                               * if the Ink should
                               * have its own
                               * GState. */
} InkDestinationInfoParams;
```

■ MSG_GEN_VIEW_SET_INK_TYPE

```
void     MSG_GEN_VIEW_SET_INK_TYPE(
         GenViewInkType inkType);   /* value of GenViewInkType */
```

This message sets the type of Ink input expected by the view's content.

Source: Unrestricted.

Destination: Any GenView object.

Parameters: *inkType* The type of Ink input expected.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_VIEW_SET_EXTENDED_INK_TYPE

```
void    MSG_GEN_VIEW_SET_EXTENDED_INK_TYPE(@stack
        Boolean createGState,
        Color  inkColor,
        word   brushSize,
        optr   destObj);
```

This message sets the extended Ink type for the GenView. The extended Ink type is stored in the vardata ATTR_GEN_VIEW_INK_DESTINATION_INFO.

Source: Unrestricted.

Destination: Any GenView object.

Parameters:

createGState	A boolean specifying whether the view should create a new GState for the Ink or not.
inkColor	A palette index to use as the color of the Ink.
brushSize	The brush size, in points, to be used for the Ink.
destObj	The optr of the destination object for Ink input to this view. After the Ink is collected, it will be sent to this object.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_VIEW_RESET_EXTENDED_INK_TYPE

```
void    MSG_GEN_VIEW_RESET_EXTENDED_INK_TYPE();
```

This message resets the extended Ink type to default values.

Source: Unrestricted.

Destination: Any GenView object.

Interception: Generally not intercepted.

9.4.8.4 Target and Focus

MSG_GEN_VIEW_UPDATE_CONTENT_TARGET_INFO

Some applications may draw some portion of their documents differently if they have the target or focus. The view will pass along the following target/focus messages: MSG_META_CONTENT_ENTER, MSG_META_CONTENT_LEAVE, MSG_META_RAW_UNIV_ENTER, MSG_META_RAW_UNIV_LEAVE, and MSG_META_CONTENT_VIEW_LOST_GADGET_EXCLUSIVE.

If you want your geode to be able to grab the focus, you must set the GVA_FOCUSABLE attribute in the *GVI_attrs* record.

■ MSG_GEN_VIEW_UPDATE_CONTENT_TARGET_INFO

```
void        MSG_GEN_VIEW_UPDATE_CONTENT_TARGET_INFO(
            ViewTargetInfo *targetInfo);
```

This message is sent to the view from **VisContentClass** contents whenever target information within the view changes. This is done so that MSG_META_GET_TARGET_EXCL may be used to retrieve target information about the portion of the target hierarchy which is within the content itself. (The UI can't directly call objects running in other threads, so this message is required to update it with the latest information.)

Source: A VisContent within the view.

Destination: The GenView associated with the caller.

Parameters: *targetInfo* A pointer to a **ViewTargetInfo** structure, shown below.

Return: Nothing.

Structures: The **ViewTargetInfo** structure has two fields, as shown:

```
typedef struct {
      optr        TR_object;  /* optr of object */
      ClassStruct *TR_class;  /* class of object */
} TargetReference;
```

```
typedef struct {
      TargetReference   VTI_target;
      TargetReference   VTI_content;
} ViewTargetInfo;
```

The *VTI_target* field contains the optr and class pointer of the object in the view that currently has the target exclusive. The *VTI_content* field contains the optr and class pointer of the view's content object.

Interception: Generally not intercepted.

9.4.8.5 Setting the Pointer Image

MSG_GEN_VIEW_SET_PTR_IMAGE

If you want to change the pointer's image while it is over your view or while the view has the focus, use MSG_GEN_VIEW_SET_PTR_IMAGE. To use this message, you must create the new pointer image in a sharable memory block, lock the block, and pass the pointer to the locked block. To reset your request (return to the default pointer), send the same message with the pointer zeroed. The pointer image structure is detailed in section 11.2.4 of chapter 11.

■ MSG_GEN_VIEW_SET_PTR_IMAGE

```
void    MSG_GEN_VIEW_SET_PTR_IMAGE(
        optr                pointerDef;     /* Optr of pointer definition */
        PtrImageLevel       level);         /* Image level of pointer */
```

This message causes the view to set the pointer image to a custom image whenever the pointer is over the view.

Source: Unrestricted.

Destination: Any GenView object.

Parameters: *pointerDef* The optr of a chunk containing the **PointerDef16** structure of the new pointer definition for the GenView (this chunk must be in a sharable memory block). Pass NullOptr for no image request. Pass the handle portion of the optr as a NullHandle and the chunk portion of the optr as a **PtrImageValue** value (see below).

level A value of **PtrImageLevel**. This should be either PIL_GADGET, to allow individual gadgets within the view to set the pointer, or PIL_WINDOW, to set the background cursor of the view. Note that if PIV_UPDATE is passed in *pointerDef*, the level parameter is not used.

Return: Nothing.

Structures: **PointerDef16** has the following structure (see the appropriate reference entry for a full description):

```
typedef struct {
    byte    PD_width;
    byte    PD_height
    sbyte   PD_hotX;
    sbyte   PD_hotY;
    byte    PD_mask[STANDARD_CURSOR_IMAGE_SIZE];
    byte    PD_image[STANDARD_CURSOR_IMAGE_SIZE];
} PointerDef16;
```

Interception: Generally not intercepted.

9.4.9 Linking Views

`GVI_horizLink, GVI_vertLink, MSG_GEN_VIEW_SEND_TO_LINKS, MSG_GEN_VIEW_SEND_TO_VLINK, MSG_GEN_VIEW_SEND_TO_HLINK, MSG_GEN_VIEW_CALL_WITHOUT_LINKS`

Views may be linked together either in the horizontal or the vertical dimension. Classed events may then be sent to the view's links, causing the events to propagate through all the links. Most scrolling events will be propagated through the links.

Each view may have one horizontal and one vertical link, though no links are required. The links are stored as optrs in the instance fields *GVI_horizLink* and *GVI_vertLink*, and links should be circular. In other words, if three views are linked in a single dimension, the links should be as follows: view one points to view two, view two points to view three, and view three points back to view one.

Views linked in the horizontal will share vertical scroll events; those linked in the vertical will share horizontal scroll events.

When one of the linked views receives a scrolling event, the event will automatically propagate to the view's links. Other events can be sent to all the linked views; by using **@record** to set up an encapsulated event, you can then send that event to the linked views with any of

MSG_GEN_VIEW_SEND_TO_LINKS, MSG_GEN_VIEW_SEND_TO_VLINK, or MSG_GEN_VIEW_SEND_TO_HLINK. If you want to send an encapsulated message to a linked view without propagating the event, use MSG_GEN_VIEW_CALL_WITHOUT_LINKS.

■ MSG_GEN_VIEW_SEND_TO_LINKS

```
void        MSG_GEN_VIEW_SEND_TO_LINKS(
            EventHandle event,      /* handle of recorded event */
            optr   originator);     /* optr of view first receiving event */
```

9.4

This message sends a recorded event to the receiving view and then to all the view's links. Any linked view will send this message upon receiving a scrolling event.

Source: A linked view when it receives a scrolling event.

Destination: The linked View sends this to itself; it will then dispatch the passed event to itself and all views linked to it.

Parameters: *event* The EventHandle of the recorded event to be dispatched to all linked views. Typically this is the scrolling event received by the originator.

originator The optr of the GenView that originally received the scrolling event and sent this message out.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_VIEW_SEND_TO_VLINK

```
void        MSG_GEN_VIEW_SEND_TO_VLINK(
            EventHandle event,      /* handle of recorded event */
            optr   originator);     /* optr of view first receiving event */
```

This message sends a recorded event to the receiving view and then to all the view's vertical links. It works the same as MSG_GEN_VIEW_SEND_TO_LINKS except that it restricts itself to vertically-linked views.

Source: A linked view when it receives a scrolling event, or a handler for MSG_GEN_VIEW_SEND_TO_LINKS.

Destination: The linked View sends this to itself; it will then dispatch the passed event to itself and all views linked to it.

Parameters: *event* The EventHandle of the recorded event to be dispatched to all vertically linked views. Typically this is the scrolling event received by the originator.

originator The optr of the GenView that originally received the scrolling event and sent this message out.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_VIEW_SEND_TO_HLINK

```
void      MSG_GEN_VIEW_SEND_TO_HLINK(
          EventHandle event,     /* handle of recorded event */
          optr   originator);    /* optr of view first receiving event */
```

This message sends a recorded event to the receiving view, then to the view's horizontal links. It works the same as MSG_GEN_VIEW_SEND_TO_LINKS except that it restricts itself to horizontally-linked views.

Source: A linked view when it receives a scrolling event, or a handler for MSG_GEN_VIEW_SEND_TO_LINKS.

Destination: The linked View sends this to itself; it will then dispatch the passed event to itself and all views linked to it.

Parameters: *event* The EventHandle of the recorded event to be dispatched to all horizontally linked views. Typically this is the scrolling event received by the originator.

originator The optr of the GenView that originally received the scrolling event and sent this message out.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_VIEW_CALL_WITHOUT_LINKS

```
void      MSG_GEN_VIEW_CALL_WITHOUT_LINKS(
          EventHandle        event,          /* handle of recorded event */
          MessageFlags       messageFlags);   /* flags normally sent with
                                            * classed event */
```

This message sends the recorded event to the view and ensures the event does not get passed on to any of the recipient's links. Because the recorded event is sent to only one GenView, it is called immediately and therefore can have return values or pass pointers.

Source: Unrestricted.

Destination: Any linked GenView object

Parameters: *event* The EventHandle of the recorded event to be sent to only the recipient GenView.

messageFlags A record of **MessageFlags** for the recorded event. If the event should return values, be sure to pass the flag MF_CALL in this record.

Return: Nothing.

Interception: Generally not intercepted.

9.4.10 Setting the Content

`MSG_GEN_VIEW_SET_CONTENT, MSG_GEN_VIEW_GET_CONTENT`

The *GVI_content* field in the GenView instance data determines what object is designated as its content. By default, an application's Process object acts as the content. Because the output field defaults to the Process object, you do not have to explicitly set it unless you plan on using a GenContent or VisContent.

Though not usually used when the Process is the content, it is possible to change the content during execution with MSG_GEN_VIEW_SET_CONTENT. Along with this message you must pass the optr of the object that will become the new content. When the view receives this message, it will shut down the current content just as if the view were being shut down. Then it will initialize the new content as if the view were opening for the first time.

To retrieve the optr of the current content object of a view, send the view MSG_GEN_VIEW_GET_CONTENT.

■ MSG_GEN_VIEW_SET_CONTENT

```
void      MSG_GEN_VIEW_SET_CONTENT(
          optr   content);
```

This message sets the view's *GVI_content* instance data field to a new optr. The current content will receive messages as if the view were shutting down, and the new content will receive messages as if the view were being created anew. (See section 9.4.1 on page 525.)

Source: Unrestricted.

Destination: Any GenView object.

Parameters: *content* The optr of the new content object.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_VIEW_GET_CONTENT

```
optr    MSG_GEN_VIEW_GET_CONTENT();
```

This message returns the optr of the object currently set as the view's content.

Source: Unrestricted.

Destination: Any GenView object.

Parameters: None.

Return: The optr currently set as the GenView's content.

Interception: Generally not intercepted.

9.4.11 Internal Utilities

MSG_GEN_VIEW_DETERMINE_VIS_PARENT

Some messages supported by the view may be used as general utilities. Most of these, however, will never be needed by applications.

■ MSG_GEN_VIEW_DETERMINE_VIS_PARENT

```
optr    MSG_GEN_VIEW_DETERMINE_VIS_PARENT(
        optr   child);
```

This message returns the optr of the visible parent for a GenView's child object. This is used by the GenView's children during its specific-UI building and should not be needed by applications.

Source: A generic child of the GenView.

Destination: The GenView being specifically built.

Parameters: *child* The optr of the child to get the visible parent of.

Return: The optr of the visible parent.

Interception: Should not be intercepted.

■ MSG_GEN_VIEW_SEND_NOTIFICATION

```
void      MSG_GEN_VIEW_SEND_NOTIFICATION();
```

This message is part of the GenViewControl mechanism and is used almost exclusively by the GenView. It sends the special notification from the GenView to the controller, if needed.

Source: Part of GenView control mechanism.

Destination: Any GenView object.

Interception: Generally not intercepted.

■ MSG_GEN_VIEW_SET_CONTROLLED_ATTRS

```
void      MSG_GEN_VIEW_SET_CONTROLLED_ATTRS(
          GenViewControlAttrs controlAttrs,
          word                scaleFactor);
```

This message sets attributes controlled by the GenViewControl.

Source: Typically a GenViewControl object, automatically.

Destination: A controlled GenView object.

Parameters: *controlAttrs* A record of **GenViewControlAttrs** flags indicating the attributes to be set.

scaleFactor The new scale factor to be set for the view.

Return: Nothing.

Interception: Generally not intercepted.

9.5 The GenViewControl

GenViewControlClass works with GenView objects to provide a number of features to your user interface. It is a subclass of **GenControlClass** and works with the GenToolControl to implement tool bars and tool boxes for the view.

For the controller to interact with a GenView, the view must have the GVA_CONTROLLED flag set in its *GVI_attrs* field. The GenViewControl will interact with all views having this flag set.

The GenViewControl features and tools are shown in Code Display 9-6. All GenViewControl objects must be placed on the MGCNLT_ACTIVE_LIST and GAGCNLT_SELF_LOAD_OPTIONS GCN lists. The view itself should be on the GAGCNLT_CONTROLLED_GEN_VIEW_OBJECTS list.

Code Display 9-6 GenViewControl Features and Tools

```
/* The GenViewControl supports the following features. */

typedef         WordFlags GVCFeatures;
#define GVCF_MAIN_100            0x4000  /* sets scale to 100% from View menu */
#define GVCF_MAIN_SCALE_TO_FIT   0x2000  /* allows scale-to-fit in View menu */
#define GVCF_ZOOM_IN             0x1000  /* zooms in on the document */
#define GVCF_ZOOM_OUT            0x0800  /* zooms out on the document */
#define GVCF_REDUCE              0x0400  /* reduces the view’s scale factor */
#define GVCF_100                 0x0200  /* sets scale to 100% in View submenu */
#define GVCF_ENLARGE             0x0100  /* enlarges the view’s scale factor */
#define GVCF_BIG_ENLARGE         0x0080  /* enlarges the scale factor twice */
#define GVCF_SCALE_TO_FIT        0x0040  /* scales the view to fit the document */
#define GVCF_ADJUST_ASPECT_RATIO 0x0020  /* adjusts scale for aspect ratio */
#define GVCF_APPLY_TO_ALL        0x0010  /* applies to all active views */
#define GVCF_SHOW_HORIZONTAL     0x0008  /* shows/hides the horizontal scroller */
#define GVCF_SHOW_VERTICAL       0x0004  /* shows/hides the vertical scroller */
#define GVCF_CUSTOM_SCALE        0x0002  /* allows custom scaling */
#define GVCF_REDRAW              0x0001  /* refreshes the view window */

/* The GenViewControl provides the following tools. */
typedef         WordFlags GVCToolboxFeatures;
#define GVCTF_100                0x1000  /* sets the scale factor to 100% */
#define GVCTF_SCALE_TO_FIT       0x0800  /* scales the view to fit the document */
#define GVCTF_ZOOM_IN            0x0400  /* zooms in; enlarges the scale factor */
#define GVCTF_ZOOM_OUT           0x0200  /* zooms out; reduces the scale factor */
#define GVCTF_REDRAW             0x0100  /* refreshes the view window */
#define GVCTF_PAGE_LEFT          0x0080  /* scrolls left/back one page width */
#define GVCTF_PAGE_RIGHT         0x0040  /* scrolls right/forward one page width */
#define GVCTF_PAGE_UP            0x0020  /* scrolls up/back one page height */
#define GVCTF_PAGE_DOWN          0x0010  /* scrolls down/forward one page height */
#define GVCTF_ADJUST_ASPECT_RATIO 0x0008 /* adjusts scale for aspect ratio */
#define GVCTF_APPLY_TO_ALL       0x0004  /* applies to all active views */
#define GVCTF_SHOW_HORIZONTAL    0x0002  /* shows/hides the horizontal scroller */
#define GVCTF_SHOW_VERTICAL      0x0001  /* shows/hides the vertical scroller */
```

```
        /* Default features and tools */
#define GVC_DEFAULT_FEATURES    (GVCF_MAIN_100 | GVCF_MAIN_SCALE_TO_FIT |
                                 GVCF_REDUCE | GCVF_ENLARGE | GVCF_BIG_ENLARGE |
                                 GVCF_SCALE_TO_FIT | GVCF_ADJUST_ASPECT_RATIO |
                                 GVCF_APPLY_TO_ALL | GVCF_SHOW_HORIZONTAL |
                                 GVCF_SHOW_VERTICAL | GVCF_CUSTOM_SCALE)

#define GVC_DEFAULT_TOOLBOX_FEATURES (GVCTF_100 | GVCTF_ZOOM_IN | GVCTF_ZOOM_OUT)

#define GVC_SUGGESTED_SIMPLE_FEATURES (GVCF_MAIN_100 | GVCF_MAIN_SCALE_TO_FIT |
                                 GVCF_ZOOM_IN | GVCF_ZOOM_OUT)

#define GVC_SUGGESTED_INTRODUCTORY_FEATURES (GVCF_MAIN_100 | GVCF_ZOOM_IN |
                                 GVCF_ZOOM_OUT)

#define GVC_SUGGESTED_BEGINNING_FEATURES (GVC_SUGGESTED_INTRODUCTORY_FEATURES |
                                 GVCF_MAIN_SCALE_TO_FIT)
```

9.5.1 GenViewControl Instance Data

In addition to the normal instance data of **GenControlClass**, **GenViewControlClass** has four other fields. These are shown in Code Display 9-7 along with their default values.

Code Display 9-7 GenViewControl Instance Data

```
/* The GenViewControl has four instance fields in addition to those inherited
 * from GenControlClass. You may wish to set these to determine the controller’s
 * initial configuration, but typically you will not need to. */

        /* Constants used for scaling boundaries */
#define DEFAULT_ZOOM_MINIMUM     25
#define DEFAULT_ZOOM_MAZIMUM    200

        /* GVCI_minZoom determines the minimum percentage zoom allowed. */
        /* GVCI_maxZoom indicates the maximum zoom percentage allowed. */
        /* GVCI_scale indicates the current view scale factor setting. */
        /* GVCI_attrs indicates which attributes the controller should
         * implement on startup. */
```

```
    @instance word      GVCI_minZoom =  DEFAULT_ZOOM_MINIMUM;
    @instance word      GVCI_maxZoom =  DEFAULT_ZOOM_MAXIMUM;
    @instance word      GVCI_scale =    100;
    @instance GenViewControlAttrs GVCI_attrs =  (GVCA_SHOW_HORIZONTAL |
                                                 GVCA_SHOW_VERTICAL |
                                                 GVCA_APPLY_TO_ALL);

        /* Possible GenViewControlAttrs flags:
         *      GVCA_ADJUST_ASPECT_RATIO        GVCA_APPLY_TO_ALL
         *      GVCA_SHOW_HORIZONTAL            GVCA_SHOW_VERTICAL */

    @default GI_attrs = (@default | GA_KBD_SEARCH_PATH);
                                /* This adds the view controller to the default
                                 * keyboard accelerator search path. */
```

9.5.2 Notification Received

When a view's attributes change and the controller will be affected, the view sends notification to the controller. The type of notification handled by the controller is NT_VIEW_STATE_CHANGE. This notification type will carry with it a block containing a structure of type **NotifyViewStateChange**, shown in Code Display 9-8.

The **NotifyViewStateChange** structure contains much more information about the view than just those features affected by the controller. Additional information includes background color, increment, document bounds, Ink information, origin, etc. If you want to add functionality to the GenViewControl, you can set up additional UI gadgetry with the **GenControlClass** attribute ATTR_GEN_CONTROL_APP_UI and have it change and handle the appropriate attributes.

Code Display 9-8 The NotifyViewStateChange Structure

```
/* This structure is passed to the GenViewControl object when a feature of the
 * active view changes. It is passed with notification NT_VIEW_STATE_CHANGE. */

typedef struct {
    PointDWFixed        NVSC_origin;            /* absolute origin of the view */
    RectDWord           NVSC_docBounds;         /* document bounds of the view */
    PointDWord          NVSC_increment;         /* the view's increment value */
    PointWWFixed        NVSC_scaleFactor;       /* view’s current scale factor */
    ColorQuad           NVSC_color;             /* view’s background color */
    GenViewAttrs        NVSC_attrs;             /* view’s GVI_attrs record */
    GenViewDimensionAttrs       NVSC_horizAttrs;/* view’s GVI_horizAttrs record */
    GenViewDimensionAttrs       NVSC_vertAttrs; /* view’s GVI_vertAttrs record */
    GenViewInkType      NVSC_inkType;           /* view’s GVI_inkType value */
  /* The following four fields are used internally by the GenViewControl. */
    XYSize              NVSC_contentSize;
    XYSize              NVSC_contentScreenSize;
    PointDWord          NVSC_originRelative;
    PointDWord          NVSC_documentSize;
} NotifyViewStateChange;
```

9.5.3 GenViewControl Example

An example of the use of a GenViewControl is shown in Code Display 9-9. This example is built on the Hello World application, and only the differences between Hello World and the example are shown. As an exercise, try adding a GenToolControl to Hello World as well.

Code Display 9-9 An Example of GenViewControl

```
/* This display builds on the Hello World sample application. You should make the
 * changes shown here to that program, compile it, and run it to get a solid feel
 * for how the GenViewControl works and the features it provides.
 * Only the alterations to Hello World are shown here; the basic code can be found
 * in “The Source File and Source Code” on page 116 of
 * “First Steps: Hello World,” Chapter 4 of the Concepts Book. */

/* HelloApp changes
 * Add a new GCN list type to the application object, and add the HelloViewControl
 * object to the list. This list is typical of system-provided controllers. */

@object GenApplicationClass HelloApp = {
    GI_visMoniker = list { @HelloTextMoniker };
    GI_comp = @HelloPrimary;
    gcnList(MANUFACTURER_ID_GEOWORKS, GAGCNLT_WINDOWS) = @HelloPrimary;
    gcnList(MANUFACTURER_ID_GEOWORKS, GAGCNLT_SELF_LOAD_OPTIONS) =       /* add */
                                                        @HelloViewControl;
}
```

```
/* HelloPrimary changes
 * Add a new menu to the Primary’s list of children. This new menu is called
 * HelloViewMenu and is declared below. */

@object GenPrimaryClass HelloPrimary = {
    GI_visMoniker = “Hello Sample Application”;          /* unchanged */
    GI_comp = @HelloView, @HelloMenu, @HelloViewMenu;    /* add @HelloViewMenu */
    ATTR_GEN_DISPLAY_NOT_MINIMIZABLE;                    /* unchanged */
    HINT_DISPLAY_SIZE_WINDOW_AS_DESIRED;                 /* unchanged */
}

/* HelloViewMenu declaration
 * Declare a new menu, HelloViewMenu. The GenViewControl object will add its
 * features to this menu. */

@object GenInteractionClass HelloViewMenu = {
    GII_visibility = GIV_POPUP;  /* This makes the GenInteraction a menu */
    GI_comp = HelloViewControl;  /* Add the GenViewControl as the menu’s child */
    ATTR_GEN_INTERACTION_GROUP_TYPE = (GIGT_VIEW_MENU);
                                 /* Give this menu the default characteristics
                                  * of the standard View menu. */
}

/* HelloViewControl declaration
 * Declare the GenViewControl object, HelloViewControl. This needs no extra
 * settings; we will use all the defaults. */

@object GenViewControlClass HelloViewControl = {
}

/* HelloView changes
 * The only change necessary to the GenView itself is to mark it controlled.
 * This entails setting the GVA_CONTROLLED flag in GVI_attrs. */

@object GenViewClass HelloView = {
    GVI_attrs = @default | GVA_CONTROLLED;       /* set the controlled attr */
    GVI_horizAttrs = @default | GVDA_SCROLLABLE | GVDA_NO_LARGER_THAN_CONTENT;
                                                 /* unchanged */
    GVI_vertAttrs = @default | GVDA_SCROLLABLE | GVDA_NO_LARGER_THAN_CONTENT;
                                                 /* unchanged */
    GVI_content = process;                       /* unchanged */
}
```

9.5

9.5.4 Messages Handled

MSG_GEN_VIEW_CONTROL_SET_ATTRS,
MSG_GEN_VIEW_CONTROL_SET_MINIMUM_SCALE_FACTOR,
MSG_GEN_VIEW_CONTROL_SET_MAXIMUM_SCALE_FACTOR

GenViewControlClass handles several messages that allow its characteristics to be set or modified at run-time. It also has several other internal messages which are not documented here.

■ MSG_GEN_VIEW_CONTROL_SET_ATTRS

```
void        MSG_GEN_VIEW_CONTROL_SET_ATTRS(
            GenViewControlAttrs attrsToSet,
            GenViewControlAttrs attrsToClear);
```

This message sets the GenViewControlAttrs in GVCI_attrs. Attributes noted in both parameters will be turned off for the controller.

Source: Unrestricted.

Destination: Any GenViewControl object.

Parameters: *attrsToSet* Attributes to be turned on should be set.

attrsToClear Attributes to be turned off should be set.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_VIEW_CONTROL_SET_MINIMUM_SCALE_FACTOR

```
void        MSG_GEN_VIEW_CONTROL_SET_MINIMUM_SCALE_FACTOR(
            word   minimumScaleFactor);
```

This message sets the view controller’s minimum allowable scale factor.

Source: Unrestricted.

Destination: Any GenViewControl object.

Parameters: *minimumScaleFactor* The new minimum scale factor (a percentage).

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_VIEW_CONTROL_SET_MAXIMUM_SCALE_FACTOR

```
void      MSG_GEN_VIEW_CONTROL_SET_MAXIMUM_SCALE_FACTOR(
           word  maximumScaleFactor);
```

This message sets the maximum settable scale factor for the view controller.

Source: Unrestricted.

Destination: Any GenViewControl object.

Parameters: *maximumScaleFactor*
The maximum settable scale factor (as a percent).

Return: Nothing.

Interception: Generally not intercepted.

9.5

Objects

The Text Objects

10

Text support is one of the integral and most important pieces of the GEOS system software. Many text formatting, editing, publishing, and advanced features are built into the system; therefore, many applications will never need to deal directly with these issues. The text features of GeoWrite, for example, are almost entirely implemented within the text object library.

If your geode will display text, receive text input, or provide any type of text formatting features, you should use one of the text objects, either GenText or VisText. (If you only need a single-line non-editable text field, you may be able to use a simple GenGlyph instead.)

The GenText object is primarily a user interface object, designed to communicate directly with the user and accept keyboard input. The VisText object is a much more versatile (and complex) object designed not only to display text but provide complex formatting and WYSIWYG word processing operations.

Both objects make use of common functionality defined within the Text Object Library. Therefore, it is important to be familiar with the features and standards of that object library before using either GenText or VisText. Additionally, if you plan on using your text object to perform any complex style sheets, you should be familiar with the chunk array mechanisms, as these are widely employed in complex text objects.

10.1 The Text Objects

Many applications will require some form of typed input from the user. Some will provide text formatting, word processing, or WYSIWYG desktop publishing features, and nearly all will have some text-based information displayed on the screen. Each of these features can be provided easily with either GenText or VisText objects. To use either of these objects, you should make sure to include the text library in your geode.

Both the VisText and GenText objects have many common features. Under the hood, both the VisText and GenText objects contain similar instance data, and both are capable of handling messages from the text object library. Which object you use depends on what role you need your text object to assume.

10.1.1 Which Object Should I Use?

The VisText object is the more complex of the two objects. In general, a VisText object may exhibit any of the many features present within the text object library; the GenText object may only exhibit a subset of this behavior. The GenText object is not a low-level object, however; in most cases, **GenTextClass** may use most of the messages provided with **VisTextClass**. It is only in more complex text operations (such as drawing text to regions or incorporating complex graphics into text) that a GenText object is not as useful.

Typically, geodes that want full-featured word processing will use a VisText object (typically within a scrollable GenView and a VisContent). Simple text editing and input (within a database field, for example) are normally done with an editable GenText object. For other cases, it is up to you. There are advantages to either approach. GenText is easier to use, but VisText is more versatile.

Typically, the makeup of your user interface will decide which text object is more appropriate. GenText objects should be attached to the generic tree; VisText objects should instead be attached to the visible tree. Most of the features listed below are described with respect to VisText objects; in most cases, however, they are equally applicable to the GenText.

The main functional difference between a VisText and a GenText object is the ability of VisText to use what is known as the *large* text model. The large model allows a text object to define flow regions, provide for text to be stored within a VM file, and offers several other rather complex features. This behavior is essential for providing complex text formatting features. It is usually not required for most text needs, however.

The GenText object does not support this text model. In cases where you feel you need to use functionality of the large model, you should use a VisText object within a visible tree. Luckily, most of these capabilities are automatic when using a VisText object. (You will rarely set up a VisText object already using this model, for instance; instead, whenever the VisText is forced to use operations requiring the large model, the application will force it to convert itself.)

10.1.2 How this Chapter is Organized

This chapter is organized in a special fashion. Not including this section, the contents of this chapter are:

10.2 A description of some of the more powerful features of the text objects.

10.3 A complete list of the features defined within the text object library, for use with both GenText and VisText.

10.4 An in-depth description of the common ways to change text and attributes within a text object. These are the most commonly used operations for text objects.

10.5 A description in detail of the VisText object.

10.6 A description in detail of the GenText object.

10.7 A description of the controllers specifically tailored for use with the text objects.

If you are using a simple GenText object for getting and setting text, you will probably only need to read the in-depth description of how to alter text and the GenText section. If you need other functionality, you should probably read this chapter straight through.

10.2 General Text Features

The Text objects provide a powerful tool for complex text formatting. This section describes some of the more powerful features of these text objects.

10.2.1 Input Management and Filters

A large portion of the text library is devoted to input management and interaction with the UI and the user. Almost every input and display function for text is implemented within the text library, relieving your applications from containing often complex code. In almost all cases, you will not need to know how input is transferred to your text objects.

10.2 The text library handles all keyboard and mouse input to text fields, interacting intelligently with the UI to process input events. (This includes pen input supported by the input manager and the pen input mechanisms.) Selection with the mouse and keyboard is handled automatically, as are quick-transfer operations.

Because the text objects understand the focus and target hierarchies of the UI, they can interact directly with the controller objects. Applications using the text objects don't even have to know when a user has applied a new style, changed the paragraph spacing, or set the text color. The user can select text, call up the Style menu, and apply a new style without using a line of application-specific code. If you wish such behavior to happen automatically, you should mark your text object GA_TARGETABLE in its *GI_attrs* instance field.

GEOS also provides filters to text objects; these filters allow the text object to selectively intercept and process text as it is being added to a text object (usually a GenText), either from direct keyboard typing or from a more complex operation (such as a clipboard paste). Filters allow the text object to accept or reject text on a number of bases (such as character by character).

10.2.2 Text Flow Through Regions

The text library also allows a continuous stream of text to flow through various connected regions. These regions may be of arbitrary size, shape, and location, and each region occupies one spot in the text object's region list. The text library, once the region list is created, takes care of all text flowing and rippling from one region to the next. GenText objects may not implement regions, though; ideally their geometry should be left up to the specific UI.

Applications can easily set up regions using the graphics system's path code, then add the regions to the text object's region list. These two simple steps can result in powerful publishing features such as flowing text around graphics and flowing text through arbitrarily-shaped regions. Unusual flow patterns are illustrated in Figure 10-1.

10.2.3 Style Sheets

Many word processors and publishing programs today implement *styles*. A style is a set of character and paragraph attributes typically associated with a specific name; the user chooses the attributes for the style and then can

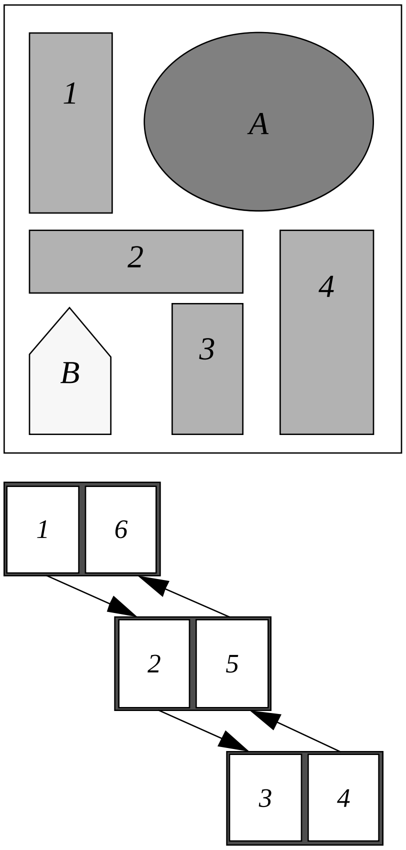


Figure 10-1 *Sample Text Flows*

Arbitrarily-Shaped Regions

Text can flow from one arbitrarily-shaped region to another. If the text object's region list were [1, 2, B, 3, 4], then text would flow from the top of region 1 to the bottom of region 4 in that order. If region A were to have text independent of the other regions, it would have to be in a different VisText object in the same visible object tree.

Specialized Layouts

Text can flow across pages if consecutive regions are on different pages. Thus, an application can set up flows that automatically arrange pages for printing pamphlets or books.

apply that style to any selected text by choosing the associated name. A *style sheet* is a set of styles. Styles can be (and normally are) based on other styles.

Normally, the definition and use of styles is left entirely to the application. In GEOS, however, the text library provides a controller object for styles and style sheets, so if your application will use style sheets, it can do so without needing complex code.

10.2.4 Undo

For complex text formatting, the ability to "undo" any changes is nearly a necessity. The GEOS text library provides a single-level undo that will work with all text-edit objects. This undo may be implemented by either the user or the application.

10.2.5 General Import and Export

Through the Impex Library, any application using a text object can import from and export to many popular word processor formats. You don't have to go through the work of creating your own format translator for each supported format; any format supported by the Impex Library will automatically be supported by your application. (See "Impex Library," Chapter 16.) In addition, the Impex Library is designed so users can add new translators to the importer/exporter; these translators are then accessible to your application without recompiling or upgrading.

10.2.6 Geometry Management

The GenText object automatically will interact with its parent generic object (usually a GenInteraction or GenPrimary) to provide the proper sizing behavior. GenText objects automatically implement their own scrolling when necessary, adding horizontal or vertical scrollers as the need arises.

VisText, however, does not provide automatic scrolling. The VisText may only exist as a node in a visible object tree under a VisContent; the content (in concert with its parent GenView) will provide whatever scrolling, scaling, and sizing that is required.

10.2.7 Embedded Graphics and Characters

Every text field with the capability to define regions may contain embedded graphics strings. Each graphic appears as a single character within the text, and graphics may be pasted in by the user.

It is also possible to flow text around graphics. The large object model provides this feature, but the regular text model does not. Thus, to flow text around graphics requires a VisText object set up for large text.

10.2.8 Search and Replace

Global search-and-replace functions are included in the text object. GEOS provides a standard search-and-replace dialog box that can be called up by any text object. The search-and-replace mechanism can work with all text objects in the system; if two applications are both using search-and-replace at the same time, only one dialog box will appear on the screen. The search and replace controller understands which application is the current target of the search operations and interacts intelligently with the active applications.

10.2.9 Spell-Checking

Like search-and-replace, spell-checking is a function of the text library and is available globally to all applications and libraries that use text objects. GEOS uses a licensed Houghton-Mifflin dictionary and spell-checking engine; this engine as well as the common UI are available to each application and library that uses a text object. The spelling checker, like the search-and-replace mechanism, interacts intelligently with the text object and the UI to ensure that only one spell-check dialog box is on the screen.

The simplest and most efficient way to include spell-checking in your application is to include an object of **SpellControlClass** in your application, linking it to the text object to spell-check. This controller is discussed in "Search and Replace and Spell-Checking" on page 706.

10.2.10 Printing

10.2

VisText objects automatically know how to print themselves when sent the proper messages. In nearly all cases where printing text is essential, a VisText will be included in a visual object tree. Then, when printing occurs, the topmost object in the tree (most likely a VisContent) will receive the print message; this message will be passed down the tree until all the visual objects (including any VisTexts) have printed themselves.

10.2.11 Text Controller Objects

As with several other major features of the GEOS user interface, the text library has associated controller classes for many features. Because text formatting has so many options and choices for the user, providing the menus, dialog boxes, and choices can be a lot of work for an application. The text controller objects, however, remove nearly all this work while allowing applications the full support they require.

Nearly every feature of the text library has one corresponding controller object. In addition, master controllers exist to manage other controller objects. Therefore, if you use a master controller, other controllers that may be added to the text object (by Geoworks) will be automatically included in your geode without changing the source code or recompiling.

Using controllers and an editable text object, you can actually create a simple but full-featured word processor without writing any code other than UI object definitions. Controllers typically work on both GenText or VisText objects; whichever text object has the target will receive updates from controllers.

10.3 The Text Object Library

Both Text Objects rely on a single framework; this framework is the Text Object Library. The Text Object Library (**tCommon.h**) contains many definitions and structures that text objects will use. Every definition and structure in this section can be used by both VisText and GenText objects.

VisTextClass is the class that utilizes most of these features directly. **GenTextClass** also utilizes almost every one of these features through **VisTextClass**. This is because **GenTextClass**, as a generic object, is manifested through **VisClass** by specific UI libraries.

The text library also has data structures and mechanisms defined to support other features that, due to time constraints, have no strict API definition in the library. To provide these "potential features" in your applications, you can directly access the text data structures and set them appropriately. (For example, automatic paragraph numbering is not directly supported, but the paragraph attributes data structures allow defining characters that get prepended to individual paragraphs.)

Many of the features are implemented and controlled automatically by GEOS. For common features (character styles and font sizes, for example), controller objects can handle almost everything. For less common features (automatic paragraph numbering, for example), your application will likely have to provide the UI and the control mechanisms necessary.

10.3.1 Character Attribute Definitions

The text object library provides a multitude of ways to shape the appearance of individual characters. Each character within a text object may even exhibit its own individual attributes, if the object is defined to use multiple character attributes. The definitions within this section show the full range of possibilities for character attributes.

10.3.1.1 The VisTextCharAttr Structure

The **VisTextCharAttr** structure provides a great variety of attributes for characters within text objects. Many of the **VisTextCharAttr** fields use graphics structures to tailor the appearance of individual characters within the text object. You may wish to review the “Graphics Environment,” Chapter 23 of the Concepts Book chapter to discover the full features of this structure.

Code Display 10-1 The VisTextCharAttr Structure

```
typedef struct {
    StyleSheetElementHeader      VTCA_meta;
    FontID                       VTCA_fontID;
    WBFixed                      VTCA_pointSize;
    TextStyle                    VTCA_textStyles;
    ColorQuad                    VTCA_color;
    sword                        VTCA_trackKerning;
    byte                         VTCA_fontWeight;
    byte                         VTCA_fontWidth;
    VisTextExtendedStyles        VTCA_extendedStyles;
    SystemDrawMask               VTCA_grayScreen;
    GraphicPattern               VTCA_pattern;
    ColorQuad                    VTCA_bgColor;
    SystemDrawMask               VTCA_bgGrayScreen;
    GraphicPattern               VTCA_bgPattern;
    byte                         VTCA_reserved[7];
} VisTextCharAttr;
```

VTCA_meta stores a **StyleSheetElementHeader**. This structure field allows the **VisTextCharAttr** structure to act as an element in an element array. For complete information on using character attribute element arrays to implement character style runs, see “Character Runs” on page 638.

VTCA_fontID stores the **FontID** of the font to be used for this character. Any current font allowed by the system is valid. There may exist up to 65536 fonts of type **FontID**.

VTCA_pointSize stores the point size of the character. This may be any fixed point value between 4 and 792.

VTCA_textStyles stores the text styles (of type **TextStyle**) to be applied to this character. For a list of text styles, see “Graphics Environment,” Chapter 23 of the Concepts Book.

◆ Objects

VTCA_color stores the current color (of type **ColorQuad**) of the foreground text character.

VTCA_trackKerning stores the kerning value (either positive or negative) for this character. This value is expressed in points and reflects how much extra space (if positive) or removal of space (if negative) to apply between this character and the next character.

VTCA_fontWeight stores the font-weight (thickness) of the character. This value is expressed in a (positive or negative) percentage of 256 and is independent of the current font in use.

VTCA_fontWidth stores the width of the font as a (positive or negative) percentage of 256 and is independent of the current font in use.

VTCA_extendedStyles stores any **VisTextExtendedStyles** (separate from the **TextStyle** in *VTCA_textStyles*) in use by this character. These styles are defined in the text object library. See Code Display 10-2 on page ◆ 597 for a list of extended styles.

VTCA_grayScreen stores the **SystemDrawMask** to use when displaying this text. This mask is applied to the text characters but not to the background.

VTCA_pattern stores the **GraphicPattern** to use when displaying this text. This pattern is applied to the text characters but not to the background.

VTCA_bgColor stores the background color to apply to the character's background (but not to the character itself).

VTCA_bgGrayScreen stores the **SystemDrawMask** to apply to the character's background.

VTCA_bgPattern stores the **GraphicPattern** to apply to the character's background.

Code Display 10-2 VisTextExtendedStyles

```
/* These flags are stored in the VisTextCharAttr entry VTCA_extendedStyles */
typedef WordFlags VisTextExtendedStyles;
#define VTES_BOXED              0x8000
#define VTES_BUTTON             0x4000
#define VTES_INDEX              0x2000
#define VTES_ALL_CAP            0x1000
```

```
#define VTES_SMALL_CAP          0x0800
#define VTES_HIDDEN             0x0400
#define VTES_CHANGE_BAR         0x0200
#define VTES_BACKGROUND_COLOR   0x0100
```

VTES_BOXED draws the character within a box (a rectangle surrounding the character). If consecutive characters are marked VTES_BOXED they will all be drawn within a single rectangular outline.

VTES_BUTTON frames the text characters within a drop shadow. Consecutive characters marked VTES_BUTTON will be framed within a single drop-shadow.

VTES_INDEX marks the character for indexing purposes. This feature is currently not implemented.

VTES_ALL_CAP indicates that the character should be capitalized in the current point size.

VTES_SMALL_CAP indicates that the character should be capitalized, but at a smaller point size.

VTES_HIDDEN indicates that this character acts as hidden text. Hidden text may exhibit special properties (such as being masked out when printing).

VTES_CHANGE_BAR is currently unimplemented.

VTES_BACKGROUND_COLOR indicates that this character exhibits a special background color.

10.3.1.2 Default Character Attributes

You may decide that you do not need your characters to exhibit the many attributes available within the **VisTextCharAttr** structure. If your text object is simple, you may want instead to use the default character attributes provided in the text object library. These default attributes supersede the use of the **VisTextCharAttr** structure. All of these attributes will fit into a single word record.

The **VisTextDefaultCharAttr** record allows your text object to exhibit three styles (underline, bold, and italic), 16 color indexes of type **Color**, eight point sizes, and 32 different fonts.

Code Display 10-3 VisTextDefaultCharAttr

```
typedef WordFlags VisTextDefaultCharAttr;
#define VTDCA_UNDERLINE 0x8000
#define VTDCA_BOLD      0x4000
#define VTDCA_ITALIC    0x2000
#define VTDCA_COLOR     0x0f00  /* Color */
#define VTDCA_SIZE      0x00e0  /* VisTextDefaultSize */
#define VTDCA_FONT      0x001f  /* VisTextDefaultFont */

#define VTDCA_COLOR_OFFSET      8
#define VTDCA_SIZE_OFFSET       5
#define VTDCA_FONT_OFFSET       0

typedef ByteEnum VisTextDefaultSize;
    #define     VTDS_8  0                /* Point size of 8 */
    #define     VTDS_9  1                /* Point size of 9 */
    #define     VTDS_10 2                /* Point size of 10 */
    #define     VTDS_12 3                /* Point size of 12 */
    #define     VTDS_14 4                /* Point size of 14 */
    #define     VTDS_18 5                /* Point size of 18 */
    #define     VTDS_24 6                /* Point size of 24 */
    #define     VTDS_36 7                /* Point size of 36 */

typedef ByteEnum VisTextDefaultFont;
    #define     VTDF_BERKELEY   0                /* Berkeley Font */
    #define     VTDF_CHICAGO    1                /* Chicago Font */
    #define     VTDF_BISON      2                /* Bison Font */
    #define     VTDF_WINDOWS    3                /* Windows Font */
    #define     VTDF_LED        4                /* LED Font */
    #define     VTDF_ROMA       5                /* Roma Font */
    #define     VTDF_UNIVERSITY 6                /* University Font */
    #define     VTDF_URW_ROMAN  7                /* URW Roman Font */
    #define     VTDF_URW_SANS   8                /* URW Sans Font */
    #define     VTDF_URW_MONO   9                /* URW Mono Font */
    #define     VTDF_URW_SYMBOLPS 10             /* URW Symbols Font */
    #define     VTDF_CENTURY_SCHOOLBOOK 11       /* Century Schoolbook Font */
```

```
#define VIS_TEXT_INITIAL_CHAR_ATTR \
        ((VTDS_12 << VTDCA_SIZE_OFFSET) || VTDF_BERKELEY)
```

10.3.1.3 Macros for Defining VisTextCharAttr Structures

The text object library also provides several macros to make the definition of the **VisTextCharAttr** structure simpler. These macros are listed in Code Display 10-4. In many cases, these macros make the definition of character style runs vastly easier.

Code Display 10-4 VisTextCharAttr Macros

```
#define CHAR_ATTR_STYLE_FONT_SIZE_STYLE_COLOR(ref, style, font, psize, tstyle, \
color) { \
    {{{ref, 0}}, style}, font, {0, psize}, tstyle, \
        {color, CF_INDEX, 0, 0}, 0, FWI_MEDIUM, FW_NORMAL, 0, SDM_100, {0}, \
        {C_WHITE, CF_INDEX, 0, 0}, SDM_0, {0}, {0, 0, 0, 0, 0, 0, 0}}

#define CHAR_ATTR_FONT_SIZE_STYLE(font, psize, tstyle) \
        CHAR_ATTR_STYLE_FONT_SIZE_STYLE_COLOR(2, CA_NULL_ELEMENT, font, \
                                        psize, tstyle, C_BLACK)

#define CHAR_ATTR_FONT_SIZE(font, psize) \
        CHAR_ATTR_STYLE_FONT_SIZE_STYLE_COLOR(2, CA_NULL_ELEMENT, font, \
                                        psize, 0, C_BLACK)
```

10.3.2 Paragraph Attribute Definitions

The text object library also provides many ways to shape the appearance of individual paragraphs. Each paragraph within a text object may exhibit its own individual attributes if the text object is set to allow multiple paragraph attributes. The definitions within this section show the full range of possibilities for paragraph attributes.

10.3.2.1 The VisTextParaAttr Structure

The **VisTextParaAttr** structure provides the largest set of attributes for use by paragraphs within text objects. These entries are listed in Code Display 10-5. Comments follow the code display.

Code Display 10-5 VisTextParaAttr

```
typedef struct {
    StyleSheetElementHeader      VTPA_meta;
    VisTextParaBorderFlags       VTPA_borderFlags;
    ColorQuad                    VTPA_borderColor;
    VisTextParaAttrAttributes    VTPA_attributes;
    word                         VTPA_leftMargin;
    word                         VTPA_rightMargin;
    word                         VTPA_paraMargin;
    BBFixedAsWord                VTPA_lineSpacing;
    word                         VTPA_leading;
    BBFixedAsWord                VTPA_spaceOnTop;
    BBFixedAsWord                VTPA_spaceOnBottom;
    ColorQuad                    VTPA_bgColor;
    byte                         VTPA_numberOfTabs;
    byte                         VTPA_borderWidth;
    byte                         VTPA_borderSpacing;
    byte                         VTPA_borderShadow;
    SystemDrawMask               VTPA_borderGrayScreen;
    SystemDrawMask               VTPA_bgGrayScreen;
    HatchPattern                 VTPA_borderHatch;
    HatchPattern                 VTPA_bgHatch;
    word                         VTPA_defaultTabs;
    word                         VTPA_startingParaNumber;
    char                         VTPA_prependChars[4];
    VisTextHyphenationInfo       VTPA_hyphenationInfo;
    VisTextKeepInfo              VTPA_keepInfo;
    VisTextDropCapInfo           VTPA_dropCapInfo;
    word                         VTPA_nextStyle;
    StandardLanguage             VTPA_language;
    byte                         VTPA_reserved[15];
} VisTextParaAttr;
```

VTPA_meta stores a **StyleSheetElementHeader**. This structure field allows the **VisTextParaAttr** structure to act as an element in an element array. For complete information on using paragraph attribute element arrays to implement paragraph style runs, see “Paragraph Runs” on page 646.

VTPA_borderFlags stores the **VisTextParaBorderFlags** to use in drawing any borders around the text paragraph.

VTPA_borderColor stores the color (of type **ColorQuad**) of the border.

VTPA_attributes stores the **VisTextParaAttrAttributes** to use on this paragraph. These attributes specify whether the paragraph uses keep-with, drop caps, automatic hyphenation, or paragraph numbering.

VTPA_leftMargin stores the left margin of the paragraph. This margin is the distance (offset in points) from the left edge of the text object.

VTPA_rightMargin stores the right margin of the paragraph. This margin is the distance (offset in points) from the right edge of the text object.

VTPA_paraMargin stores the paragraph margin. This margin is the additional distance (offset in points from the left margin) to indent the first line of a paragraph.

VTPA_lineSpacing stores the distance between lines of the paragraph. Single-spacing (a *VTPA_lineSpacing* of 1.0) is the default value.

VTPA_leading stores the leading of the paragraph. Leading is the additional space between all lines of a paragraph, independent of line spacing. This value is expressed in points.

VTPA_spaceOnTop stores the additional vertical distance (expressed in points) before the first line of the paragraph. *VTPA_spaceOnTop* plus *VTPA_spaceOnBottom* is the total vertical distance between paragraphs.

VTPA_spaceOnBottom stores the additional vertical distance (expressed in points) after the last line of the paragraph. *VTPA_spaceOnTop* plus *VTPA_spaceOnBottom* is the total vertical distance between paragraphs.

VTPA_bgColor stores the background color (of type **ColorQuad**) of the paragraph. Note this is different from the background color of an individual character (in *VTCA_bgColor*). The background color of a character, if any, takes precedence over the background color of a paragraph.

VTPA_numberOfTabs stores the number of tab stops contained in this paragraph. Tabs are added at the end of the **VisTextParaAttr** structure.

Therefore, this structure may be of variable size, depending on the number of tabs.

VTPA_borderWidth stores the width (in units of eight pixels) of the border, if any, attached to this paragraph.

VTPA_borderSpacing stores the spacing (in units of eight pixels) between the paragraph border, if any, and the paragraph itself.

VTPA_borderShadow stores the distance (in units of eight pixels) of the border shadow, if any, from the main border.

VTPA_borderGrayScreen stores the **SystemDrawMask** to use when drawing the paragraph border.

VTPA_bgGrayScreen stores the **SystemDrawMask** to use when drawing the paragraph background.

VTPA_borderHatch stores **HatchPattern** to use when drawing the border.

VTPA_bgHatch stores the **HatchPattern** to use when drawing the background.

VTPA_defaultTabs stores the spacing for default tabs.

VTPA_startingParaNumber stores the sequential number of this paragraph. This value depends on the **VisTextNumberType** in *VTPA_attributes*.

VTPA_prependChars stores the characters (up to four) to prepend to the beginning of this paragraph.

VTPA_hyphenationInfo stores the **VisTextHyphenationInfo** (see below) to use between lines of text within a paragraph.

VTPA_keepInfo stores the **VisTextKeepInfo** (see below) to use between this and other paragraphs.

VTPA_dropCapInfo stores the **VisTextDropCapInfo** (see below) to use at the start of this paragraph.

VTPA_nextStyle and *VTPA_language* represent text attributes that are currently unimplemented but will be added.

VTPA_reserved reserves fifteen bytes of data at the end of the **VisTextParaAttr** structure. These bytes are used to designate extra tabs.

Code Display 10-6 VisTextParaBorderFlags

```
typedef WordFlags VisTextParaBorderFlags;
#define VTPBF_LEFT              0x8000
#define VTPBF_TOP               0x4000
#define VTPBF_RIGHT             0x2000
#define VTPBF_BOTTOM            0x1000
#define VTPBF_DOUBLE            0x0800
#define VTPBF_DRAW_INNER_LINES  0x0400
#define VTPBF_SHADOW            0x0200
#define VTPBF_ANCHOR            0x0003 /* ShadowAnchor */

#define VTPBF_ANCHOR_OFFSET     0

typedef ByteEnum ShadowAnchor;
#define SA_TOP_LEFT     0
#define SA_TOP_RIGHT    1
#define SA_BOTTOM_LEFT  2
#define SA_BOTTOM_RIGHT 3
```

The **VisTextParaBorderFlags** specify the manner in which a border should be drawn around the paragraph.

VTPBF_LEFT, VTPBF_TOP, VTPBF_RIGHT, and VTPBF_BOTTOM all specify whether their respective side is drawn with a border. If you wish a border to entirely surround the text object, you should set all of these flags.

VTPBF_DOUBLE specifies that there should be a two-line border around this paragraph. This flag will double any lines set to be drawn with VTPBF_LEFT, VTPBF_TOP, VTPBF_RIGHT, and VTPBF_BOTTOM.

VTPBF_DRAW_INNER_LINES draws lines between bordered paragraphs. If two consecutive paragraphs are marked with this flag, a line border will be drawn between them.

VTPBF_SHADOW specifies that the border should be shadowed. The direction of shadowing is specified in the VTPBF_ANCHOR flag.

VTPBF_ANCHOR specifies the **ShadowAnchor** to use if drawing a border shadow. The **ShadowAnchor** specifies the corner of the paragraph to anchor the shadow. The shadowing effect on the paragraph will proceed from this corner towards the opposite corner.

Objects

Code Display 10-7 VisTextNumberType

```
typedef ByteEnum VisTextNumberType;
#define VTNT_NUMBER             0
#define VTNT_LETTER_UPPER_A     1
#define VTNT_LETTER_LOWER_A     2
#define VTNT_ROMAN_NUMERAL_UPPER3
#define VTNT_ROMAN_NUMERAL_LOWER4
```

The **VisTextNumberType** specifies the paragraph numbering scheme for this paragraph. For each paragraph with a numbering type, the object will number that paragraph sequentially based on the last occurrence of a paragraph with the same type. The text object stores this sequential number in the *VTPA_startingParaNumber* entry.

Code Display 10-8 VisTextParaAttrAttributes

```
typedef WordFlags VisTextParaAttrAttributes;
#define VTPAA_JUSTIFICATION             0xc000
#define VTPAA_KEEP_PARA_WITH_NEXT       0x2000
#define VTPAA_KEEP_PARA_TOGETHER        0x1000
#define VTPAA_ALLOW_AUTO_HYPHENATION    0x0800
#define VTPAA_DISABLE_WORD_WRAP         0x0400
#define VTPAA_COLUMN_BREAK_BEFORE       0x0200
#define VTPAA_PARA_NUMBER_TYPE          0x01c0
#define VTPAA_DROP_CAP                  0x0020
#define VTPAA_KEEP_LINES                0x0010

#define VTPAA_JUSTIFICATION_OFFSET      14
#define VTPAA_PARA_NUMBER_TYPE_OFFSET   5
```

VTPAA_JUSTIFICATION stores the **Justification** to use with this paragraph.

VTPAA_KEEP_PARA_WITH_NEXT ensures that this paragraph and the next one will not be split along page breaks.

VTPAA_KEEP_PARA_TOGETHER ensures that the entire paragraph will not be split along page breaks. In most cases, this will cause a paragraph capable of being split to appear at the top of the next page.

VTPAA_ALLOW_AUTO_HYPHENATION allows the paragraph to automatically hyphenate words that cross line breaks. If this flag is set, the entry *VTPA_hyphenationInfo* will store the **VisTextHyphenationInfo** to use when hyphenating words.

VTPAA_DISABLE_WORD_WRAP disables automatic word wrapping. Each line break will immediately wrap to next line without keeping words together.

VTPAA_PARA_NUMBER_TYPE stores the **VisTextParaType** for this paragraph. The actual paragraph number is stored in the *VTPA_startingParaNumber* entry. The paragraph number type in tandem with the paragraph number determines what number heading this paragraph will have.

VTPAA_DROP_CAP signifies that a this paragraph uses Drop Caps. If this flag is set, the **VisTextParaAttr** entry *VTPA_dropCapInfo* stores the **VisTextParaDropCapInfo** (see below).

VTPAA_KEEP_LINES specifies that this paragraph should make sure that beginning or ending lines of certain lengths should not be allowed to stand alone. If this flag is set, the **VisTextParaAttr** entry *VTCA_keepInfo* stores the **VisTextKeepInfo** (see below).

Code Display 10-9 VisTextHyphenationInfo

```
typedef WordFlags VisTextHyphenationInfo;
#define VTHI_HYPHEN_MAX_LINES           0xf000
#define VTHI_HYPHEN_SHORTEST_WORD       0x0f00
#define VTHI_HYPHEN_SHORTEST_PREFIX     0x00f0
#define VTHI_HYPHEN_SHORTEST_SUFFIX     0x000f

#define VTHI_HYPHEN_MAX_LINES_OFFSET            12
#define VTHI_HYPHEN_SHORTEST_WORD_OFFSET         8
#define VTHI_HYPHEN_SHORTEST_PREFIX_OFFSET       4
#define VTHI_HYPHEN_SHORTEST_SUFFIX_OFFSET       0
```

If the *VTPA_attributes* flag VTPAA_ALLOW_AUTO_HYPHENATION is set, **VisTextHyphenationInfo** stores the criteria to use when hyphenating words in *VTPA_hyphenationInfo*. If hyphenation is set, words will be automatically hyphenated using a Houghton-Mifflin engine.

◆Objects

VTHI_SHORTEST_WORD sets the shortest word-length (in character counts) to hyphenate. Words shorter than this character count will be wrapped to the next line.

VTHI_SHORTEST_PREFIX and VTHI_SHORTEST_SUFFIX set the shortest prefix and suffix to leave on a line after hyphenation. Prefixes or suffixes shorter than this character count will be tied to the main word and wrapped, if necessary.

Code Display 10-10 VisTextKeepInfo

```
typedef ByteFlags VisTextKeepInfo;
#define VTKI_TOP_LINES          0xf0
#define VTKI_BOTTOM_LINES       0x0f

#define VTKI_TOP_LINES_OFFSET   4
#define VTKI_BOTTOM_LINES_OFFSET 0
```

If the *VTPA_attributes* flag VTPAA_KEEP_LINES is set, **VisTextKeepInfo** specifies the number of lines at the beginning and end of a paragraph that should not remain across page breaks. This prevents “widows” and “orphans”—single lines either at the end of one page or the beginning of another.

VTKI_TOP_LINES specifies the minimum number of lines at the beginning of a paragraph to be considered able to stand alone and not kept part of the main paragraph along a page break.

VTKI_BOTTOM_LINES specifies the minimum number of lines at the end of a paragraph to be considered able to stand alone and not kept part of the main paragraph along a page break.

For example, if VTKI_BOTTOM_LINES is set to three, then single or double lines will not be able to stand on their own at the top of a page, and either the orphans will be kept with the main paragraph or additional line(s) will be brought over from the preceding page to pad the next page.

Code Display 10-11 VisTextDropCapInfo

```
typedef WordFlags VisTextDropCapInfo;
#define VTDCI_CHAR_COUNT        0xf000
#define VTDCI_LINE_COUNT        0x0f00
#define VTDCI_POSITION          0x00f0

#define VTDCI_CHAR_COUNT_OFFSET         12
#define VTDCI_LINE_COUNT_OFFSET         8
#define VTDCI_POSITION_OFFSET           4
```

10.3

If the *VTPA_attributes* flag VTPAA_DROP_CAP is set, **VisTextDropCapInfo** specifies the criteria to use when implementing Drop Caps. Drop Caps are currently not implemented in GEOS though they will be supported for later releases.

VTDCI_CHAR_COUNT sets the number of characters to capitalize at the start of the paragraph before resuming normal capitalization.

VTDCI_LINE_COUNT sets the number of lines to capitalize at the start of a paragraph before resuming normal capitalization.

Code Display 10-12 Tabs

```
typedef ByteEnum TabLeader;
#define TL_NONE         0
#define TL_DOT          1
#define TL_LINE         2
#define TL_BULLET       3
#define TL_GRAY_LINE    4

typedef ByteEnum TabType;
#define TT_LEFT         0
#define TT_CENTER       1
#define TT_RIGHT        2
#define TT_ANCHORED     3

typedef ByteFlags TabAttributes;
#define TA_LEADER       0x1c            /* TabLeader */
#define TA_TYPE         0x03            /* TabType */

#define TA_LEADER_OFFSET2
#define TA_TYPE_OFFSET  0
```

```
typedef struct {
    word                T_position;
    TabAttributes       T_attr;         /* TabAttributes */
    SystemDrawMask      T_grayScreen;
    byte                T_lineWidth;
    byte                T_lineSpacing;
    word                T_anchor;
} Tab;

typedef struct {
    VisTextParaAttr     VTMPA_paraAttr;
    Tab                 VTMPA_tabs[VIS_TEXT_MAX_TABS];
} VisTextMaxParaAttr;
```

The **Tab** structure allows your paragraph to add custom tab stops to a paragraph. These tabs are in addition to any default tab stops that are defined in *VTPA_defaultTabs*. You may add up to 25 custom tabs to each paragraph. These tabs are added at the end of the **VisTextParaAttr** structure. Therefore, different paragraphs may have paragraph attributes of different lengths depending on the number of custom tabs in use by that paragraph.

10.3.2.2 Default Paragraph Attributes

You may decide that you do not need many of the features within the **VisTextParaAttr** structure in your paragraphs. If your text object is simple, you may want to use a set of default paragraph attributes provided in the text object library. These default attributes supersede the use of the **VisTextParaAttr** structure. All of these attributes will fit into a single word record of type **VisTextDefaultParaAttr**.

The **VisTextDefaultParaAttr** structure allows your text object to exhibit a **Justification**, a subset of default tabs, and left, right, and paragraph margins.

Code Display 10-13 VisTextDefaultParaAttr

```
typedef WordFlags VisTextDefaultParaAttr;
#define VTDPA_JUSTIFICATION      0xc000 /* Justification */
#define VTDPA_DEFAULT_TABS       0x3000 /* VisTextDefaultTab */
#define VTDPA_LEFT_MARGIN        0x0f00 /* In units of half-inches */
#define VTDPA_PARA_MARGIN        0x00f0 /* In units of half-inches */
#define VTDPA_RIGHT_MARGIN       0x000f /* In units of half-inches */

#define VTDPA_JUSTIFICATION_OFFSET       14
#define VTDPA_DEFAULT_TABS_OFFSET        12
#define VTDPA_LEFT_MARGIN_OFFSET         8
#define VTDPA_PARA_MARGIN_OFFSET         4
#define VTDPA_RIGHT_MARGIN_OFFSET        0

#define VIS_TEXT_INITIAL_PARA_ATTR ( (0*2) << VTDPA_LEFT_MARGIN_OFFSET) | \
                                 ( (0*2) << VTDPA_PARA_MARGIN_OFFSET) | \
                                 ( (0*2) << VTDPA_RIGHT_MARGIN_OFFSET) | \
                                 (VTDDT_INCH << VTDPA_DEFAULT_TABS_OFFSET) | \
                                 (J_LEFT << VTDPA_JUSTIFICATION_OFFSET)

typedef ByteEnum VisTextDefaultDefaultTab;
#define VTDDT_NONE               0
#define VTDDT_HALF_INCH          1
#define VTDDT_INCH               2
#define VTDDT_CENTIMETER         3
```

The **VisTextDefaultDefaultTab** type is solely for use within a **VisTextDefaultParaAttr** record. The default tab stops provide either no tab stops (VTDDT_NONE) or tab stops every half-inch, centimeter, or inch. Note that no other application-defined tabs are allowed when using the default paragraph attributes.

10.3.2.3 Macros for Defining VisTextParaAttr Structures

The text object library also provides a macro to make the definition of the **VisTextParaAttr** structure simpler; there are also a few macros to aid in setting up justification and tab structures. These macros are listed in Code Display 10-14. In many cases, these macros make the definition of paragraph style runs vastly easier.

Code Display 10-14 VisTextParaAttr Macros

```
#define PARA_ATTR_STYLE_JUST_LEFT_RIGHT_PARA(ref, style, just, left, right, para)\
{ \
        {{{ref, 0}}, style}, 0, {C_BLACK, CF_INDEX, 0, 0}, \
        just << VTPAA_JUSTIFICATION_OFFSET, (left)*PIXELS_PER_INCH, \
        (right)*PIXELS_PER_INCH, (para)*PIXELS_PER_INCH, \
        1<<8, 0, 0, 0, {C_WHITE, CF_INDEX, 0, 0}, \
        0, 1*8, 2*8, 1*8, SDM_100, SDM_0, {0}, {0}, \
        PIXELS_PER_INCH/2*8, VIS_TEXT_DEFAULT_STARTING_NUMBER, ““, 0, 0, 0,
        CA_NULL_ELEMENT, SL_ENGLISH, {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}}

#define DEF_PARA_ATTR_JUST_TABS(just, tabs) \
        (( (0*2) << VTDPA_LEFT_MARGIN_OFFSET ) | \
         ( (0*2) << VTDPA_PARA_MARGIN_OFFSET ) | \
         ( (0*2) << VTDPA_RIGHT_MARGIN_OFFSET ) | \
         ( (tabs) << VTDPA_DEFAULT_TABS_OFFSET ) | \
         ( (just) << VTDPA_JUSTIFICATION_OFFSET ))

#define DEF_PARA_ATTR_CENTER DEF_PARA_ATTR_JUST_TABS(J_CENTER, VTDDT_INCH)

#define DEF_PARA_ATTR_RIGHT DEF_PARA_ATTR_JUST_TABS(J_RIGHT, VTDDT_INCH)
```

10.3.3 Storage Flags

As you have seen, the Text Object Library provides a multitude of character and paragraph attributes to use in the display of text. The library also provides a number of different subsets of attributes and ways to store these attributes.

A simple text object may only need a 16-bit record (such as **VisTextDefaultCharAttr** or **VisTextDefaultParaAttr**) to store its text attributes; a highly complex text object may need a chunk array of several 30+ byte elements. The manner in which these attributes are stored is specified in the **VisTextStorageFlags** of the text object.

Code Display 10-15 VisTextStorageFlags

```
typedef ByteFlags VisTextStorageFlags;
#define VTSF_LARGE                0x80
#define VTSF_MULTIPLE_CHAR_ATTRS 0x40
#define VTSF_MULTIPLE_PARA_ATTRS 0x20
#define VTSF_TYPES                0x10
#define VTSF_GRAPHICS             0x08
#define VTSF_DEFAULT_CHAR_ATTR   0x04
#define VTSF_DEFAULT_PARA_ATTR   0x02
#define VTSF_STYLES               0x01
```

VTSF_LARGE

This flag states that this text object is using the large model for its data. In this case, the rest of the settings in **VisTextStorageFlags** are ignored. The large model is a highly complex text formatting tool; you should avoid it entirely until you are familiar with other aspects of the text object.

VTSF_MULTIPLE_CHAR_ATTRS

This flag specifies that the text may display multiple character attributes. Each character may then display its own character attributes independent of other characters. If this flag is set, VTSF_DEFAULT_CHAR_ATTRS must be not set.

VTSF_MULTIPLE_PARA_ATTRS

This flag specifies that the text may display multiple paragraph attributes. Each paragraph may then display its own attributes independent of other paragraphs. If this flag is set, VTSF_DEFAULT_PARA_ATTRS must not be set.

VTSF_TYPES

This flag specifies that this text object contains a types run.

VTSF_GRAPHICS

This flag specifies that this text object contains a graphics run.

VTSF_DEFAULT_CHAR_ATTRS

This flag specifies that this text object only uses a set of provided default character attributes. These default attributes are represented by a record of type **VisTextDefaultCharAttrs**. If this flag is not set, the text object will use the larger **VisTextCharAttr** structure to store its character attributes instead.

VTSF_DEFAULT_PARA_ATTRS

This flag specifies that this text object only uses a set of provided default paragraph attributes. These default attributes are represented by a record of type **VisTextDefaultParaAttrs**. If this flag is not set, the text object will use the larger **VisTextParaAttr** structure to store its paragraph attribute.

VTSF_STYLES

This flag specifies that this text object contains a styles run.

10.4 Text Object Chunks

Each text object contains several instance fields that may reference outside chunks. Most non-large text objects will reference the following chunks:

- A text chunk containing the string of text for that object.
- A lines/fields chunk containing internal information about the length and position of lines and tab fields.
- A character attributes chunk, containing either a single **VisTextCharAttr** structure or (for multiple character attributes) a list of **VisTextCharAttr** structures.
- A paragraph attributes chunk, containing either a single **VisTextParaAttr** structure or (for multiple paragraph attributes) a list of **VisTextParaAttr** structures.
- For multiple character attributes, a chunk of character attribute runs, referencing the list of **VisTextCharAttr** structures above.
- For multiple paragraph attributes, a chunk of paragraph attribute runs, referencing the list of **VisTextParaAttr** structures above.

The Text Object Library provides many other features, but these are the most often-used and difficult to master. You should have a complete grasp of these mechanisms before using either VisText or GenText. To make their use easier, both objects use these features in essentially the same manner.

10.4.1 The Text

The most important data associated with a text object is the text itself. The text of all non-large Text Objects is stored within a single chunk; this chunk is located in the same resource as the object itself. Text objects contain an instance field with a ChunkHandle to this text chunk. (In GenText objects, this is the instance field *GTXI_text*; in VisText objects, this is *VTI_text*.)

Unless your text object uses the complex large model, it stores all of its text within this chunk. You may set initial text for your text object to appear within its instance data. The text object library automatically handles keyboard input from the user and translates that into characters displayed within the text object, altering the contents of the text chunk at the same time.

The text within the chunk is represented by a null-terminated character string. Each character of the text occupies a zero-based position within the text field. (The first character in the text field is at position zero.) These character positions are useful for setting character and paragraph attributes, marking the current selection, and marking the insertion point for new text.

The text object library provides a number of operations that you can perform to alter the display of text. These messages may take text from a variety of sources and include it within your text object; alternatively, you may retrieve text from your object and send it to other objects or processes.

Though the text within a non-large text object will always reside in a chunk (and may therefore be referenced with an optr) text may come from (and go to) several different sources. If you intend to retrieve text from or send text to a text object, you must know what type of source (or destination) you are dealing with.

Typically, text outside of a text object will reside in one of the following six forms. The first two cases are the most common.

- A pointer to null terminated text string.
- An optr to a chunk. (This is the same format the text object itself stores its text.)
- A handle of a global memory block.
- A handle of a VM block.

- A database item.
- A huge array.

For each operation on a text object, the text object library provides specific messages tailored to the format you are retrieving text from or transferring text to.

10.4.1.1 Text Ranges

VisTextRange, MSG_VIS_TEXT_GET_RANGE, VisTextRangeContext, MSG_VIS_TEXT_GET_TEXT_SIZE

Frequently, your application may wish to specify a range of text to act on. This range specifies the starting and ending points within the text for the relevant operation. Each of these starting and ending points is a zero-based character position.

```
typedef struct {
    dword   VTR_start;
                  /* starting character position */
    dword   VTR_end;
                  /* ending character position */
} VisTextRange;
```

To select a starting point at the first character, set *VTR_start* to zero. To select an ending point at the last character, set *VTR_end* to the special constant TEXT_ADDRESS_PAST_END.

If you want to pass the current selection as a range in any message that demands a **VisTextRange**, pass VIS_TEXT_RANGE_SELECTION to indicate that the currently selected area should be used to specify the range. Note that for some operations (such as paragraph attribute changes) the affected area may be larger than the text selection. Pass VIS_TEXT_RANGE_PARAGRAPH_SELECTION if you want the currently selected area to be used after it has been adjusted to reflect its paragraph boundaries.

You can use MSG_VIS_TEXT_GET_RANGE to return an actual **VisTextRange** of the selection (or paragraph selection). Pass this message a **VisTextRangeContext**, which specifies whether the range will include just the selection or kick out the range to its paragraph boundaries.

```
typedef WordFlags VisTextRangeContext;
#define VTRC_PARAGRAPH_CHANGE            0x8000
#define VTRC_CHAR_ATTR_CHANGE            0x4000
#define VTRC_PARA_ATTR_BORDER_CHANGE     0x2000
```

MSG_VIS_TEXT_GET_TEXT_SIZE returns the total size of the text within the text object.

■ MSG_VIS_TEXT_GET_RANGE

```
void      MSG_VIS_TEXT_GET_RANGE(
          VisTextRange        *range,
          word                context);
```

This message fills in a **VisTextRange** buffer based on the selection criteria passed in the *context* argument. The context information specifies whether the range will be used for a character attribute change (in which case the normal selection positions will be used) or whether the range will be used for a paragraph attribute change (in which case the boundaries of the selection will be kicked out to paragraph boundaries).

Source: Unrestricted.

Destination: Any text object.

Parameters: *range* Buffer to hold the range text positions returned from the message handler.

context **VisTextRangeContext**.

Return: The **VisTextRange** buffer filled in.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_GET_TEXT_SIZE

```
dword     MSG_VIS_TEXT_GET_TEXT_SIZE();
```

This message returns the current size of text.

Source: Unrestricted.

Destination: Any text object.

Return: The size of the text within the text object.

Interception: Generally not intercepted.

10.4.1.2 Replacing Text All At Once

```
MSG_VIS_TEXT_REPLACE_ALL_PTR,
MSG_VIS_TEXT_REPLACE_ALL_OPTR,
MSG_VIS_TEXT_REPLACE_ALL_BLOCK,
MSG_VIS_TEXT_REPLACE_ALL_VM_BLOCK,
MSG_VIS_TEXT_REPLACE_ALL_DB_ITEM,
MSG_VIS_TEXT_REPLACE_ALL_HUGE_ARRAY
```

You may replace the text displayed within a Text Object all at once with the following messages. Any previous text within the text object will be replaced with new text. The new text may come from any of the formats previously described. The format of this source determines which message you should use to replace the text.

MSG_VIS_TEXT_REPLACE_ALL_PTR replaces the text with text referenced by a simple pointer to a text string.

MSG_VIS_TEXT_REPLACE_ALL_OPTR replaces the text with text referenced by an optr. (This is the format that non-large text objects store their text.)

MSG_VIS_TEXT_REPLACE_ALL_BLOCK replaces the text with text residing in a global memory block.

MSG_VIS_TEXT_REPLACE_ALL_VM_BLOCK replaces the text with text residing in a VM block. Text objects that use the large model usually store their text within a VM block, so this message is useful for transferring text from one VisText object to another, for example.

MSG_VIS_TEXT_REPLACE_ALL_DB_ITEM replaces the text with text from a database item.

MSG_VIS_TEXT_REPLACE_ALL_HUGE_ARRAY replaces the text with text from a huge array.

■ MSG_VIS_TEXT_REPLACE_ALL_PTR

```
void      MSG_VIS_TEXT_REPLACE_ALL_PTR(
          const char          *text,
          word                textLen);
```

This message replaces the text string within a text object (either a VisText or a GenText object) with the text referenced by the passed pointer.

Source: Unrestricted.

Destination: Any GenText or VisText object.

Parameters: *text* A pointer to a text string.

textLen The text length (in characters) or zero if null-terminated.

Return: Nothing. The Text object's text is replaced.

Interception: Generally not intercepted.

10.4

■ MSG_VIS_TEXT_REPLACE_ALL_OPTR

```
void        MSG_VIS_TEXT_REPLACE_ALL_OPTR(
            optr o,
            word textLen);
```

This message replaces the text string within a text object (either a VisText or a GenText) with the text referenced by the passed optr.

Source: Unrestricted.

Destination: Any GenText or VisText object.

Parameters: *o* The optr of the chunk containing the text.

textLen Length of the text string or zero if null-terminated.

Return: Nothing. The text object's text is replaced.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_REPLACE_ALL_BLOCK

```
void        MSG_VIS_TEXT_REPLACE_ALL_BLOCK(
            word block,
            word textLen);
```

This message replaces the text string within a text object (either a VisText or a GenText) with the text within the passed data block.

Source: Unrestricted.

Destination: Any GenText or VisText object.

Parameters: *block* The handle of the data block containing the text to use in the replacement operation.

textLen The length of the text or zero if null-terminated.

Return: Nothing. The text object's text is replaced.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_REPLACE_ALL_VM_BLOCK

```
void       MSG_VIS_TEXT_REPLACE_ALL_VM_BLOCK(
           VMFileHandle       file,
           VMBlockHandle      block,
           word               textLen);
```

This message replaces the text string within a text object (either a VisText or a GenText) with the text within the passed data block.

Source: Unrestricted.

Destination: Any GenText or VisText object.

Parameters: *file* The handle of the VM file containing the text.

block The VM block handle of the block within the VM file.

textLen The length of the text or zero if null-terminated.

Return: Nothing. The text object's text is replaced.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_REPLACE_ALL_DB_ITEM

```
void       MSG_VIS_TEXT_REPLACE_ALL_DB_ITEM(
           VMFileHandle       file,
           DBGroup            group,
           DBItem             item);
```

This message replaces the text string within a text object (either a VisText or a GenText) with the text within the passed database item. The text is assumed to be null-terminated.

Source: Unrestricted.

Destination: Any GenText or VisText object.

Parameters: *file* The handle of the database item's associated VM file.

group The database group containing the text.

item The DB item containing the text.

Return: Nothing. The text object's text is replaced.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_REPLACE_ALL_HUGE_ARRAY

```
void      MSG_VIS_TEXT_REPLACE_ALL_HUGE_ARRAY(
          VMFileHandle        file,
          VMBlockHandle       hugeArrayBlock,
          word                textLen);
```

This message replaces the text string within a text object (either a VisText or a GenText) with the text within the passed HugeArray.

Source: Unrestricted.

Destination: Any GenText or VisText object.

Parameters: *file* The handle of the huge array's associated VM file.

hugeArrayBlock The VM block handle of the huge array directory block.

textLen The length of the text or zero if null-terminated.

Return: Nothing. The text object's text is replaced.

Interception: Generally not intercepted.

10.4.1.3 Replacing the Text Within a Selection

```
MSG_VIS_TEXT_REPLACE_SELECTION_PTR,
MSG_VIS_TEXT_REPLACE_SELECTION_OPTR,
MSG_VIS_TEXT_REPLACE_SELECTION_BLOCK,
MSG_VIS_TEXT_REPLACE_SELECTION_VM_BLOCK,
MSG_VIS_TEXT_REPLACE_SELECTION_DB_ITEM,
MSG_VIS_TEXT_REPLACE_SELECTION_HUGE_ARRAY
```

The text object library automatically allows text selection. By using the mouse (or other device, depending on the specific UI), a user may select any section of text and perform operations on this selected text. The following messages replace any selected text with text from one of the described sources. What message you use depends on what type of source your text comes from.

Note that if no text is selected, these messages will enter text at the current cursor position. This allows these messages either to replace selected text or to "insert" text if none is selected. You may override this insertion behavior by setting paragraph attributes. In those cases, text entered at the current position will overstrike current text.

■ MSG_VIS_TEXT_REPLACE_SELECTION_PTR

```
void    MSG_VIS_TEXT_REPLACE_SELECTION_PTR(
        const char          *text,
        word                textLen);
```

This message replaces the current selection within a text object with the text contained in the passed string pointer. If no text is currently selected, the text will be inserted at the current cursor position.

Source: Unrestricted.

Destination: Any GenText or VisText object.

Parameters: *text* The pointer to the character string.

textLen The length of the text in characters or zero, if null-terminated.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_REPLACE_SELECTION_OPTR

```
void    MSG_VIS_TEXT_REPLACE_SELECTION_OPTR(
        optr   o,
        word   textLen);
```

This message replaces the current selection within a text object with the text in the chunk specified by *o*. If no text is currently selected, the text will be inserted at the current cursor position.

Source: Unrestricted.

Destination: Any GenText or VisText object.

Parameters: *o* The optr of the text chunk.

textLen The length of the text in characters or zero, if null-terminated.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_REPLACE_SELECTION_BLOCK

```
void      MSG_VIS_TEXT_REPLACE_SELECTION_BLOCK(
          word   block,
          word   textLen);
```

This message replaces the current selection within a text object with the text contained in the passed block. If no text is currently selected, the text will be inserted at the current cursor position.

Source: Unrestricted.

Destination: Any GenText or VisText object.

Parameters: *block* Handle of the text block.

textLen The length of the text in characters or zero, if null-terminated.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_REPLACE_SELECTION_VM_BLOCK

```
void      MSG_VIS_TEXT_REPLACE_SELECTION_VM_BLOCK(
          VMFileHandle        file,
          VMBlockHandle       block,
          word                textLen);
```

This message replaces the current selection within a text object by the text contained in the passed VM block. If no text is currently selected, the text will be inserted at the current cursor position.

Source: Unrestricted.

Destination: Any GenText or VisText object.

Parameters: *file* Handle of the VM file containing the text.

block Handle of the block within the VM file.

textLen The length of the text in characters or zero, if null-terminated.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_REPLACE_SELECTION_DB_ITEM

```
void        MSG_VIS_TEXT_REPLACE_SELECTION_DB_ITEM(
            VMFileHandle        file,
            DBGroup             group,
            DBItem              item);
```

This message replaces the current selection within a text object with the text contained in the passed database item. If no text is currently selected, the text will be inserted at the current cursor position. The text is assumed to be null-terminated.

Source: Unrestricted.

Destination: Any GenText or VisText object.

Parameters:

file	The handle of the VM file containing the text.
group	The database group containing the text.
item	The database item.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_REPLACE_SELECTION_HUGE_ARRAY

```
void        MSG_VIS_TEXT_REPLACE_SELECTION_HUGE_ARRAY(
            VMFileHandle        file,
            VMBlockHandle       hugeArrayBlock,
            word                textLen);
```

This message replaces the current selection within a text object with the text contained in the passed huge array. If no text is currently selected, the text will be inserted at the current cursor position.

Source: Unrestricted.

Destination: Any GenText or VisText object.

Parameters:

file	The VM file containing the huge array.
hugeArrayBlock	The VM block handle of the huge array directory block.
textLen	The length of the text in characters or zero, if null-terminated.

Return: Nothing.

Interception: Generally not intercepted.

10.4.1.4 Appending the Text

MSG_VIS_TEXT_APPEND_PTR, MSG_VIS_TEXT_APPEND_OPTR, MSG_VIS_TEXT_APPEND_BLOCK, MSG_VIS_TEXT_APPEND_VM_BLOCK, MSG_VIS_TEXT_APPEND_DB_ITEM, MSG_VIS_TEXT_APPEND_HUGE_ARRAY

In many cases, you may not want to replace text but instead add it to the end of the current text. You may use any of the following messages to append text to your text object. Again, the new text may come from any of the formats previously described. The format of this source determines which message you should use to add the text.

■ MSG_VIS_TEXT_APPEND_PTR

```
void    MSG_VIS_TEXT_APPEND_PTR(
        const char          *text,
        word                textLen);
```

This message appends text to a text object; the text is added at the end of the current text. None of the previous text is changed.

Source: Unrestricted.

Destination: Any VisText or GenText object.

Parameters: *text* The pointer to the character string.

textLen The length of the text in characters or zero, if null-terminated.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_APPEND_OPTR

```
void    MSG_VIS_TEXT_APPEND_OPTR(
        optr   o,
        word   textLen);
```

This message adds text to a text object; the text is added at the end of the current text. None of the previous text is changed.

Source: Unrestricted.

Destination: Any VisText or GenText object.

Parameters: *o* The optr of the text chunk.

textLen The length of the text in characters or zero, if null-terminated.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_APPEND_BLOCK

```
void    MSG_VIS_TEXT_APPEND_BLOCK(
        word   block,
        word   textLen);
```

This message appends text to a text object; the text is added at the end of the current text. None of the previous text is changed.

Source: Unrestricted.

Destination: Any VisText or GenText object.

Parameters: *block* Handle of the text block.

textLen The length of the text in characters or zero, if null-terminated.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_APPEND_VM_BLOCK

```
void    MSG_VIS_TEXT_APPEND_VM_BLOCK(
        VMFileHandle        file,
        VMBlockHandle       block,
        word                textLen);
```

This message appends text to a text object; the text is added at the end of the current text. None of the previous text is changed.

Source: Unrestricted.

Destination: Any VisText or GenText object.

Parameters: *file* Handle of the VM file containing the text.

block Handle of the block within the VM file.

textLen The length of the text in characters or zero, if null-terminated.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_APPEND_DB_ITEM

```
void        MSG_VIS_TEXT_APPEND_DB_ITEM(
            VMFileHandle        file,
            DBGroup             group,
            DBItem              item);
```

This message appends text to a text object; the text is added at the end of the current text. None of the previous text is changed. The text is assumed to be null-terminated.

Source: Unrestricted.

Destination: Any VisText or GenText object.

Parameters:		
	file	The handle of the VM file containing the text.
	group	The database group containing the text.
	item	The database item.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_APPEND_HUGE_ARRAY

```
void        MSG_VIS_TEXT_APPEND_HUGE_ARRAY(
            VMFileHandle        file,
            VMBlockHandle       hugeArrayBlock,
            word                textLen);
```

This message appends text to a text object; the text is added at the end of the current text. None of the previous text is changed.

Source: Unrestricted.

Destination: Any VisText or GenText object.

Parameters:		
	file	The VM file containing the huge array.
	hugeArrayBlock	The block handle of the huge array.
	textLen	The length of the text in characters or zero, if null-terminated.

Return: Nothing.

Interception: Generally not intercepted.

10.4.1.5 Retrieving the Text

MSG_VIS_TEXT_GET_ALL_PTR, MSG_VIS_TEXT_GET_ALL_OPTR, MSG_VIS_TEXT_GET_ALL_BLOCK, MSG_VIS_TEXT_GET_ALL_VM_BLOCK, MSG_VIS_TEXT_GET_ALL_DB_ITEM, MSG_VIS_TEXT_GET_ALL_HUGE_ARRAY

You may also retrieve the text from your text object and place its text into any of the previously mentioned formats. The format of your destination determines which message you should use.

■ MSG_VIS_TEXT_GET_ALL_PTR

```
word    MSG_VIS_TEXT_GET_ALL_PTR(
        const char          *text);
```

This message retrieves the entire text of a text object and copies it to the buffer passed. The text within the text object is unchanged.

Source: Unrestricted.

Destination: Any VisText or GenText object.

Parameters: *text* The pointer to a locked or fixed buffer. The buffer must be large enough to accommodate all the text.

Return: The length of the null-terminated string not counting the null character.

text The buffer pointed to by *text* will contain the null-terminated text of the text object.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_GET_ALL_OPTR

```
word    MSG_VIS_TEXT_GET_ALL_OPTR(
        optr   o);
```

This message retrieves the entire text of a text object and copies it into the chunk specified by *o*. The text within the text object is unchanged.

Source: Unrestricted.

Destination: Any VisText or GenText object.

Parameters: *o* The optr of the chunk into which the text will be copied. Pass a valid memory handle with a null chunk handle to have the message allocate a new chunk.

Return: The chunk handle of the resized (or new) chunk. The chunk will always exist upon return and will contain at least the terminating null character.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_GET_ALL_BLOCK

```
word    MSG_VIS_TEXT_GET_ALL_BLOCK(
        word   block);
```

This message retrieves the entire text of a text object and copies it into the passed data block.

Source: Unrestricted.

Destination: Any VisText or GenText object.

Parameters: *block* Handle of the memory block into which the text will be copied. Pass a null handle to have the message allocate a new block.

Return: The handle of the resized (or new) block. The block will always exist upon return and will contain at least the terminating null character.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_GET_ALL_VM_BLOCK

```
word    MSG_VIS_TEXT_GET_ALL_VM_BLOCK(
        VMFileHandle       file,
        VMBlockHandle      block);
```

This message retrieves the entire text of a text object and copies it into the passed VM block.

Source: Unrestricted.

Destination: Any VisText or GenText object.

Parameters: *file* The VM file handle of the VM file.

block The VM block handle of the VM block into which the text will be copied. Pass a null handle to have the message allocate a new VM block in the file.

Return: The VM block handle of the resized (or new) VM block. The VM block will always exist (assuming a valid VM file was passed), and it will contain at least the terminating null character.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_GET_ALL_DB_ITEM

```
DBGroupAndItem MSG_VIS_TEXT_GET_ALL_DB_ITEM(
          VMFileHandle       file,
          DBGroup            group,
          DBItem             item);
```

This message retrieves the entire text of a text object and copies it into the passed database item.

Source: Unrestricted.

Destination: Any VisText or GenText object.

Parameters: *file* The VM file handle of the VM file.

group The database group number (may be DB_UNGROUPED—if so, a new group will be returned as part of the return value).

item The item number of the database item into which the text will be copied. Pass zero to allocate a new database item in the specified *group*.

Return: The **DBGroupAndItem** representing the resized (or new) DB item into which the text was copied. The item will always exist (assuming a valid VM file was specified) upon return, and it will contain at least the terminating null character.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_GET_ALL_HUGE_ARRAY

```
word      MSG_VIS_TEXT_GET_ALL_HUGE_ARRAY(
          VMFileHandle       file,
          VMBlockHandle      hugeArrayBlock);
```

This message retrieves the entire text of a text object and copies it into the passed huge array block.

Source: Unrestricted.

Destination: Any VisText or GenText object.

Parameters: *file* The VM file handle of the VM file containing the huge array.

hugeArrayBlock The VM block handle of the huge array. Pass a null handle to have the message allocate a new huge array.

Return: The VM block handle of the first block of the resized (or new) huge array. The huge array will always exist upon return (assuming a valid VM file was specified), and it will contain at least the terminating null character.

Interception: Generally not intercepted.

10.4.1.6 Retrieving the Text Within a Selection

```
MSG_VIS_TEXT_GET_SELECTION_PTR,
MSG_VIS_TEXT_GET_SELECTION_OPTR,
MSG_VIS_TEXT_GET_SELECTION_BLOCK,
MSG_VIS_TEXT_GET_SELECTION_VM_BLOCK,
MSG_VIS_TEXT_GET_SELECTION_DB_ITEM,
MSG_VIS_TEXT_GET_SELECTION_HUGE_ARRAY
```

To retrieve the text within the current selection, you may send any of the following messages to your text object. If no text is currently selected no text will be retrieved.

■ MSG_VIS_TEXT_GET_SELECTION_PTR

```
word    MSG_VIS_TEXT_GET_SELECTION_PTR(
        const char          *text);
```

This message retrieves the currently selected text and stores it in the character string referenced by the passed pointer. The text remains selected and intact in the original text object. The passed buffer must be large enough to accommodate the text.

Source: Unrestricted.

Destination: Any VisText or GenText object.

Parameters: *text* The pointer to the character string.

Return: The length of the string not including the terminating null character.

text The buffer pointed to by *text* will contain the null-terminated character string.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_GET_SELECTION_OPTR

```
word    MSG_VIS_TEXT_GET_SELECTION_OPTR(
        optr   o);
```

This message retrieves the currently selected text and copies it into the passed chunk. The text remains selected and intact in the original text object.

Source: Unrestricted.

Destination: Any VisText or GenText object.

Parameters: *o* The optr of the text chunk into which the text will be copied. Pass a valid memory handle and a null chunk handle to have the message allocate a new chunk in the give block.

Return: The chunk handle of the resized (or new) chunk containing the text. The chunk will contain at least the terminating null character.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_GET_SELECTION_BLOCK

```
word    MSG_VIS_TEXT_GET_SELECTION_BLOCK(
        word   block);
```

This message retrieves the currently selected text and copies it into the passed data block. The text remains selected and intact in the original text object.

Source: Unrestricted.

Destination: Any VisText or GenText object.

Parameters: *block* Handle of the text block to place the text into. Pass a null handle to have the message allocate a new memory block.

Return: The memory handle of the resized (or new) block containing the selected text. The block will contain at least the terminating null character.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_GET_SELECTION_VM_BLOCK

```
word      MSG_VIS_TEXT_GET_SELECTION_VM_BLOCK(
          VMFileHandle        file,
          VMBlockHandle       block);
```

This message retrieves the currently selected text and copies it into the passed VM block. The text remains selected and intact in the original text object.

10.4

Source: Unrestricted.

Destination: Any VisText or GenText object.

Parameters: *file* The VM file handle of the VM file.

block The VM block handle of the VM block into which the text will be copied. Pass a null block handle to have the message allocate a new VM block in the file.

Return: The VM block handle of the resized (or new) VM block. The VM block will contain at least the terminating null character.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_GET_SELECTION_DB_ITEM

```
DBGroupAndItem MSG_VIS_TEXT_GET_SELECTION_DB_ITEM(
          VMFileHandle        file,
          DBGroup             group,
          DBItem              item);
```

This message retrieves the currently selected text and copies it into the given database item. The text remains selected and intact in the original text object.

Source: Unrestricted.

Destination: Any VisText or GenText object.

Parameters: *file* The VM file handle of the VM file.

group The group number of the passed item (if DB_UNGROUPED, a group will be determined).

item The item number of the database item into which the text will be copied. Pass zero to have the message allocate a new database item in the specified group and file.

Return: The **DBGroupAndItem** representing the resized (or new) database item. The item will contain at least the terminating null character.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_GET_SELECTION_HUGE_ARRAY

```
word    MSG_VIS_TEXT_GET_SELECTION_HUGE_ARRAY(
        VMFileHandle        file,
        VMBlockHandle       hugeArrayBlock);
```

This message retrieves the currently selected text and copies it into the passed huge array. The text remains selected and intact in the original text object.

Source: Unrestricted.

Destination: Any VisText or GenText object.

Parameters: *file* The VM file handle of the VM file.

hugeArrayBlock The VM block handle of the huge array into which the text will be copied. Pass a null handle to have the message allocate a new huge array in the passed VM file.

Return: The VM block handle of the resized (or new) huge array block. The huge array will contain at least the terminating null character.

Interception: Generally not intercepted.

10.4.1.7 Other Operations on the Text

`MSG_VIS_TEXT_DELETE_ALL, MSG_VIS_TEXT_DELETE_SELECTION`

To delete the entire contents of a text object, send it MSG_VIS_TEXT_DELETE_ALL. This message will also resize the text's chunk to its minimum size. To delete only the current selection, send the text object MSG_VIS_TEXT_DELETE_SELECTION. No text will be selected after this message is sent.

■ MSG_VIS_TEXT_DELETE_ALL

```
void    MSG_VIS_TEXT_DELETE_ALL();
```

This message deletes the entire contents of a text object's text chunk. The chunk will be resized to zero.

Source: Unrestricted.

Destination: Any VisText or GenText object.

Parameters: None.

Return: Nothing

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_DELETE_SELECTION

10.4

```
void    MSG_VIS_TEXT_DELETE_SELECTION();
```

This message deletes the currently selected text. The text after the deletion will be automatically repositioned, and the text chunk will be resized.

Source: Unrestricted.

Destination: Any VisText or GenText object.

Parameters: None.

Return: Nothing.

Interception: Generally not intercepted.

10.4.1.8 Marking the Text Selection

```
MSG_VIS_TEXT_GET_SELECTION_RANGE,
MAG_VIS_TEXT_SELECT_RANGE,
MSG_VIS_TEXT_SELECT_RANGE_SMALL, MSG_VIS_TEXT_SELECT_ALL,
MSG_VIS_TEXT_SELECT_START, MSG_VIS_TEXT_SELECT_END,
MSG_VIS_TEXT_SELECT_RELATIVE
```

MSG_VIS_TEXT_GET_SELECTION_RANGE returns the **VisTextRange** of the current selection.

MSG_VIS_TEXT_SELECT_RANGE selects the selected area of text to the passed range. This message can be used with both large and small model text objects.

MSG_VIS_TEXT_SELECT_RANGE_SMALL selects a range of text. The message must pass the **VisTextRange** to “select.” This message only works with non-large text objects. Any previous selection will be deselected.

MSG_VIS_TEXT_SELECT_ALL selects the recipient’s entire text.

MSG_VIS_TEXT_SELECT_START selects the start of the text with a **VisTextRange** of zero length. This places the cursor at the beginning of the text object and deselects any previous selection.

MSG_VIS_TEXT_SELECT_END selects the end of the text with a **VisTextRange** of zero length. This places the cursor at the end of the text object and deselects any previous selection.

MSG_VIS_TEXT_SELECT_RELATIVE moves the cursor from its current position to a new location relative to the old position. You may pass this message a relative selection range to move the cursor and select text at the new position.

■ MSG_VIS_TEXT_GET_SELECTION_RANGE

```
void    MSG_VIS_TEXT_GET_SELECTION_RANGE(
        VisTextRange        *vtr);
```

This message returns the range of the text object's current selection. You must pass this message a **VisTextRange** buffer for the message to fill in with the selection range.

Source: Unrestricted.

Destination: Any VisText or GenText object.

Parameters: *vtr* A pointer to a **VisTextRange** structure to fill in with the selection range.

Return: The **VisTextRange** buffer will be filled in.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_SELECT_RANGE

```
void    MSG_VIS_TEXT_SELECT_RANGE(@stack
        dword  end
        dword  start);
```

This message creates a selection for a text object. The message must pass the starting and ending character positions to mark as the text object's selection. Any previous selection will be deselected. Note that this message uses dword offsets into the text, and can therefore be used with LARGE model text objects.

Source: Unrestricted.

Destination: Any VisText or GenText object.

Parameters: *start* The zero-based character position of the start of the selection.

end The zero-based character position of the end of the selection. This must be equal to or larger than the start position.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_SELECT_RANGE_SMALL

```
void    MSG_VIS_TEXT_SELECT_RANGE_SMALL(
        word   start,
        word   end);
```

This message creates a selection for a text object. The message must pass the starting and ending character positions to mark as the text object's selection. Any previous selection will be deselected.

Source: Unrestricted.

Destination: Any VisText or GenText object.

Parameters: *start* The zero-based character position of the start of the selection.

end The zero-based character position of the end of the selection. This must be equal to or larger than the start position.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_SELECT_ALL

```
void    MSG_VIS_TEXT_SELECT_ALL();
```

This message selects the entire text of a text object as its selection.

Source: Unrestricted.

Destination: Any VisText or GenText object.

Parameters: None.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_SELECT_START

```
void        MSG_VIS_TEXT_SELECT_START();
```

This message places the cursor at the start of the text. Any current selection will be deselected.

Source: Unrestricted.

Destination: Any VisText or GenText object.

Parameters: None.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_SELECT_END

```
void        MSG_VIS_TEXT_SELECT_END();
```

This message places the cursor at the end of the text. Any current selection will be deselected.

Source: Unrestricted.

Destination: Any VisText or GenText object.

Parameters: None.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_SELECT_RELATIVE

```
void        MSG_VIS_TEXT_SELECT_RELATIVE(
            word   newStart,
            word   newEnd);
```

This message moves a selection from the current cursor position to a position relative to that position.

Source: Unrestricted.

Destination: Any VisText or GenText object.

Parameters: *newStart* The number of character positions from the current cursor position to start the new selection.

newEnd number of character positions from the current cursor position to end the new selection. This value must be equal to or greater than *newStart*.

Return: Nothing.

Interception: Generally not intercepted.

10.4.2 Lines and Fields

Each text object contains a chunk storing its line and field information. The line and field information is internal; you should not alter it.

10.4.3 Character Runs

The character attributes of your text object may be singular or multiple. Singular character attributes have one common set of character attributes for use by all characters within a text object. Multiple character attributes store their characteristics in "runs" which allow separate characters in the same object to show different character attributes.

10.4.3.1 Singular Character Attributes

In many cases, your text object will need only one set of attributes for all characters. In those cases, the text object's character attributes instance field (*VTI_charAttrRuns* for VisText, ATTR_GEN_TEXT_DEFAULT_CHAR_ATTR or ATTR_GEN_TEXT_CHAR_ATTR for GenText) will store a single set of attributes. This word of data will contain either a 16-bit record or a ChunkHandle to a **VisTextCharAttr** structure. Those attributes will be exhibited by all characters in that text object.

Unless your **VisTextStorageFlags** specify VTSF_MULTIPLE_CHAR_ATTRS, your character attributes will be singular. (For GenText, this field is set automatically if using the singular attributes shown above.) All text within the text object will exhibit the same characteristics; also, any changes to the character attributes will affect every character in the entire text. If you need your text object to allow different characters to show different attributes (such as in a word processor), see "Multiple Character Attributes" on page 640.

If your character attributes are singular, you have two choices: use a default set of characteristics or use a more complex (and complete) set of characteristics. For most simple generic text objects, the default set offers enough variety to accomplish most goals. If you need other character styles not offered in the default set, you will have to use the complete set of character attributes.

To use the default character attributes with a VisText, specify VTSF_DEFAULT_CHAR_ATTRS in your **VisTextStorageFlags**. The default set uses the **VisTextDefaultCharAttr** record to store the character's attributes. This record offers a simple collection of character fonts, point sizes, and other attributes (see below). If you need other character styles not found in this record, you should not set VTSF_DEFAULT_CHAR_ATTRS. The text object will automatically use the **VisTextCharAttr** structure.

By default, text objects using the default character attributes will be set to a font of VTDF_BERKELEY, a point size of VTDS_12, and a text color of C_BLACK.

If you do not use the default character attributes record, you must use the **VisTextCharAttr** structure to store your text object's character attributes. (In this case, do not set the VTSF_DEFAULT_CHAR_ATTRS flag in **VisTextStorageFlags**.) The text object's character attributes instance field will contain a ChunkHandle to this structure instead of the **VisTextDefaultCharAttr** record (see Code Display 10-16).

Code Display 10-16 Setting Character Attributes

```
/* This example shows how to set the default character attributes. */

@object GenTextClass MyTextObject = {
    GTXI_text = “”;
    ATTR_GEN_TEXT_DEFAULT_CHAR_ATTR = (VTDCA_UNDERLINE |
                                (VTDS_18 << VTDCA_SIZE_OFFSET) | VTDF_BERKELEY);
                /* This sets the default character attributes record (of type
                 * VisTextDefaultCharAttr) to use underlined Berkeley font in
                 * point size 18. */
}

/* This example shows how to use non-default singular character attributes. */
```

```
@object GenTextClass MyTextObject = {
    GTXI_text = “”;
    ATTR_GEN_TEXT_CHAR_ATTR = (ChunkHandle) @MyTextCharAttrs;
                /* This is a chunk handle to the chunk holding the text
                 * attributes (of type VisTextCharAttr). */
}

@chunk VisTextCharAttr MyCharAttrs =
    CHAR_ATTR_FONT_SIZE_STYLE(FID_DTC_URW_ROMAN, 32, TS_UNDERLINE);
                /* This chunk holds the record that will determine the character
                 * attributes of all the GenText’s text. The text will be
                 * 32-point Roman, underlined. */
```

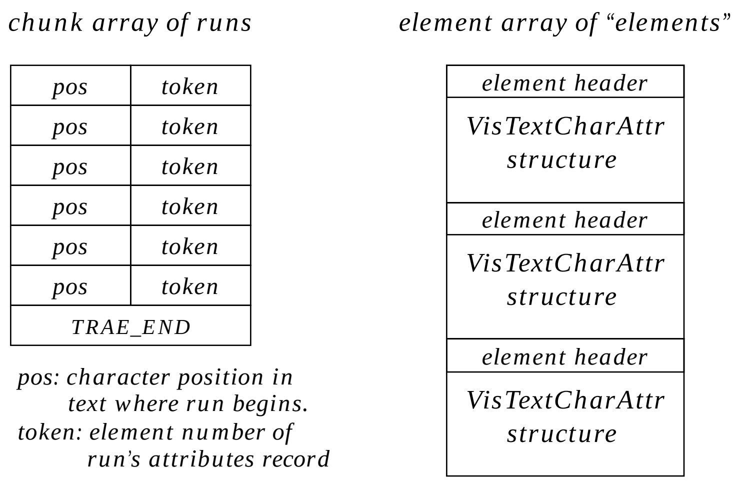
10.4.3.2 Multiple Character Attributes

To allow a VisText object to exhibit multiple character attributes, you should set VTSF_MULTIPLE_CHAR_ATTRS and clear VTSF_DEFAULT_CHAR_ATTR flag in the object’s **VisTextStorageFlags** (stored in *VTI_storageFlags*). For GenText, you should set ATTR_GEN_TEXT_MULTIPLE_CHAR_ATTR_RUNS. Any multiple character attribute object will not be able to use any of the default character attributes of type **VisTextDefaultCharAttr**.

Multiple character attributes allow your object’s individual characters to exhibit different attributes. One character may be bold and italic; another may be underlined and in a different font. Any character may exhibit any of the attributes allowed within the **VisTextCharAttr** structure.

Multiple attributes are stored as lists of **VisTextCharAttr** structures within element arrays. (If you are not familiar with element arrays, see “Local Memory,” Chapter 16 of the Concepts Book.) Because each character may exhibit different characteristics, the list of multiple character attributes may be of varying size, depending on the breadth of choice within a particular text object.

Character attributes are specified by “runs.” Each character run is used until another run is encountered, at which point the new character attributes are used. Runs are defined by a character position within a chunk array and an associated token element. At the character position, the token corresponds to a **VisTextCharAttr** element. The character attributes specified in this



pos: character position in text where run begins.
token: element number of run’s attributes record

Figure 10-2 *Structure of a Style Run*

Each entry in the chunk array of runs contains a zero-based character position and an index token identifying the character attributes record. Each element contains the VisTextCharAttr structure to use for the current run.

VisTextCharAttr structure are used until the character position of the next **VisTextCharAttr** token.

If, for example, a GenText object contains the text “Initial Text Here” with the word “Text” underlined, the chunk array of elements would look as shown in Figure 10-3.

Code Display 10-17 Setting Multiple Character Attribute Runs

```
@object GenTextClass MultipleCharAttrsObject = {
    ATTR_GEN_TEXT_MULTIPLE_CHAR_ATTR_RUNS = (ChunkHandle) @MyMultCharAttrs;
    GTXI_text = “Initial Text Here”;
}
#define MCA_PLAIN       0
#define MCA_UNDERLINE   1

CHAR_ATTR_ELEMENT_ARRAY_HEADER CharAttrElements = {
    CHAR_ATTR_FONT_SIZE_STYLE(FID_DTC_URW_ROMAN, 12, 0),          /* Element 0 */
    CHAR_ATTR_FONT_SIZE_STYLE(FID_DTC_URW_ROMAN, 12, TS_UNDERLINE)/* Element 1 */
};
```

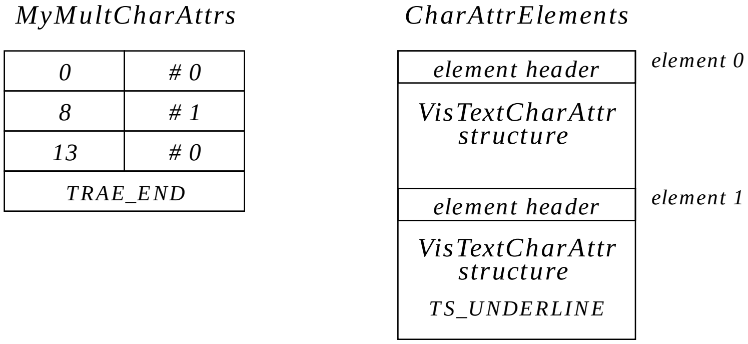


Figure 10-3 *Example of a Character Run*

GEN_TEXT_MULTIPLE_CHAR_ATTR_RUNS points to the chunk array containing the character runs (in this case, MyMultCharAttrs). This chunk array refers to the element array containing the actual character styles of each run (in this case, CharAttrElements). This example shows three style runs of two different styles.

```
RUN_ARRAY_HEADER(CharAttrElements) MyMultCharAttrs = {
    TRAE_ABS(0, MCA_PLAIN),             /* Element 0 */
    TRAE_ABS(8, MCA_UNDERLINE),         /* Element 1 */
    TRAE_END
};
```

10.4.3.3 Changing Character Attributes

MSG_VIS_TEXT_GET_CHAR_ATTR,VisTextCharAttrFlags, VisTextCharAttrDiffs, VisTextGetAttrFlags, MSG_VIS_TEXT_SET_CHAR_ATTR_BY_DEFAULT, MSG_VIS_TEXT_SET_CHAR_ATTR, MSG_VIS_TEXT_SET_FONT_ID, MSG_VIS_TEXT_SET_POINT_SIZE, MSG_VIS_TEXT_SET_TEXT_STYLE

There may be cases in which you would like to change the attributes exhibited by certain characters in your Text object. In most cases, you can do this most easily by including one of the controllers mentioned at the end of this chapter. The controllers leave any underlying work up to the specific controller implementation.

If you wish to manually change these character attributes, however, there are several messages to retrieve and set character attributes.

■ MSG_VIS_TEXT_GET_CHAR_ATTR

```
word      MSG_VIS_TEXT_GET_CHAR_ATTR(@stack
          VisTextGetAttrFlags    flags,
          VisTextCharAttrDiffs   *diffs,
          VisTextCharAttr        *attrs,
          dword                  rangeEnd,
          dword                  rangeStart);
```

This message returns a buffer filled in with the **VisTextCharAttr** attributes of the given range of text. If VTGAF_MERGE_WITH_PASSED is passed in the **VisTextGetAttrFlags**, then the passed **VisTextCharAttr** structure will be merged with the range of text that this message is sent to. (If this flag is not passed, any information initially in that buffer will be ignored.)

Over the passed range, different characters may exhibit different attributes; a pointer to a **VisTextCharAttrDiffs** structure is also passed to store information about attributes that are different across the range of text. This structure contains a list of **VisTextCharAttrFlags** specifying what attributes are multiply present. The structure also contains the bitfields of the **TextStyle** and **VisTextExtendedStyles** present over the range of text.

Possible **VisTextCharAttrFlags**:

VTCAF_MULTIPLE_FONT_IDS
VTCAF_MULTIPLE_POINT_SIZES
VTCAF_MULTIPLE_COLORS
VTCAF_MULTIPLE_GRAY_SCREENS
VTCAF_MULTIPLE_PATTERNS
VTCAF_MULTIPLE_TRACK_KERNINGS
VTCAF_MULTIPLE_FONT_WEIGHTS
VTCAF_MULTIPLE_FONT_WIDTHS
VTCAF_MULTIPLE_BG_COLORS
VTCAF_MULTIPLE_BG_GRAY_SCREENS
VTCAF_MULTIPLE_BG_PATTERNS
VTCAF_MULTIPLE_STYLES

Structures:

```
typedef struct {
      VisTextCharAttrFlags     VTCAD_diffs;
      VisTextExtendedStyles    VTCAD_extendedStyles;
      TextStyle                VTCAD_textStyles;
      byte                     VTCAD_unused;
} VisTextCharAttrDiffs;
```

Source: Unrestricted.

Destination: Any text object.

Parameters: *flags* VTGAF_MERGE_WITH_PASSED to merge the retrieved text attributes with the text in the passed range.

diffs Pointer to a **VisTextCharAttrDiffs** structure to store attribute differences.

attrs Pointer to a **VisTextCharAttr** buffer to store the retrieved character attributes. If VTGAF_MERGE_WITH_PASSED is passed in flags, this buffer initially contains attributes to match against the retrieved attributes.

rangeEnd End of the range (character position).

rangeStart Beginning of the range (character position).

Return: The token of the specific character attribute run (word value) if the text object is storing runs of **VisTextCharAttr** structures. The *attrs* and *diffs* buffers are also filled with their relevant information.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_SET_CHAR_ATTR_BY_DEFAULT

```
void     MSG_VIS_TEXT_SET_CHAR_ATTR_BY_DEFAULT(@stack
         VisTextDefaultCharAttr     defCharAttrs,
         dword                      rangeEnd,
         dword                      rangeStart);
```

This message sets the character attributes passed in **VisTextDefaultCharAttr** over the specified range of the text object. If the text object is not in “default” character attribute mode (i.e. it is storing runs of **VisTextCharAttr** structures) it will translate the default attributes into their matching **VisTextCharAttr** attributes.

Source: Unrestricted.

Destination: Any Text object.

Parameters: *defCharAttrs* **VisTextDefaultCharAttr** record to set over the passed range of text.

rangeEnd End of the range (Character position).

rangeStart Start of the range (Character position).

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_SET_CHAR_ATTR

```
void      MSG_VIS_TEXT_SET_CHAR_ATTR(@stack
          VisTextCharAttr     *attrs,
          dword               rangeEnd,
          dword               rangeStart);
```

This message sets the character attributes passed in the **VisTextCharAttr** buffer over the specified range of the text object. If the text object is in “default” character attribute mode, it will force the text object to begin storing runs of **VisTextCharAttr** structures.

Source: Unrestricted.

Destination: Any Text object.

Parameters: *attrs* Pointer to a **VisTextCharAttr** buffer.

rangeEnd End of the range (Character position).

rangeStart Start of the range (Character position).

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_SET_FONT_ID

```
@importMessage MetaTextMessages, void MSG_VIS_TEXT_SET_FONT_ID(@stack
          FontID              fid,
          dword               rangeEnd,
          dword               rangeStart);
```

This message sets the passed **FontID** over the specified range of the text object. If the text object currently displays its values in “default” character attributes, it will force the text object to use character attribute runs to store its **VisTextCharAttr** structures.

Source: Unrestricted.

Destination: Any Text object.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_SET_POINT_SIZE

```
@importMessage MetaTextMessages, void MSG_VIS_TEXT_SET_POINT_SIZE(@stack
          WWFixedAsDWord      pointSize,
          dword               rangeEnd,
          dword               rangeStart);
```

This message sets the passed point size over the specified range of the text object. If the text object currently displays its values in “default” character attributes, it will force the text object to use character attribute runs to store its **VisTextCharAttr** structures.

Source: Unrestricted.

Destination: Any Text object.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_SET_TEXT_STYLE

```
@importMessage MetaTextMessages, void MSG_VIS_TEXT_SET_TEXT_STYLE(@stack
          word                extBitsToClear,
          word                extBitsToSet,
          word                styleBitsToClear,
          word                styleBitsToSet,
          dword               rangeEnd,
          dword               rangeStart);
```

This message sets the passed text styles over the specified range of the text object. You must specify the specific style bits to set and clear, both for the *VTCA_styles* and the *VTCA_extendedStyles* entries.

If the text object currently displays its values in “default” character attributes, it will force the text object to use character attribute runs to store its **VisTextCharAttr** structures.

Source: Unrestricted.

Destination: Any Text object.

Interception: Generally not intercepted.

10.4.4 Paragraph Runs

The paragraph attributes of your text object may also be singular or multiple. Singular paragraph attributes use one common set of characteristics for all the text in the text object. Multiple paragraph attributes store their characteristics in “runs” which allow separate paragraphs in the same object to use different attributes.

10.4.4.1 Singular Paragraph Attributes

You may only need your text object to exhibit a singular set of paragraph attributes. In this case, the text object’s paragraph attributes instance field (*VTI_paraAttrRuns* for VisText, GEN_TEXT_DEFAULT_PARA_ATTR or GEN_TEXT_PARA_ATTR for GenText) will store a single set of attributes. This word of data will contain either a 16-bit record or a ChunkHandle to a chunk of more complex attributes.

Unless your **VisTextStorageFlags** (in *VTI_storageFlags*) specify VTSF_MULTIPLE_PARA_ATTRS, your paragraph attributes will be singular. (This flag will be set automatically for GenTexts using the above attributes.) Initially, all text within the text object will exhibit these same characteristics; also, any changes to the text object’s paragraph attributes will affect the entire text. If your text object should allow different paragraphs to exhibit different attributes, see “Multiple Paragraph Attributes” on page 648.

If your paragraph attributes are singular, you have two choices: use a default set of characteristics or use a more complex (and complete) set of characteristics. What you use depends on the variety of choices you wish your application to have over the display of its text. For most simple generic text objects, the default set offers enough variety to accomplish most goals.

To use the default paragraph attributes, specify VTSF_DEFAULT_PARA_ATTRS in the text object’s *VTI_storageFlags* field (for VisText) or set ATTR_GEN_TEXT_DEFAULT_PARA_ATTR (for GenText). The default set uses the **VisTextDefaultParaAttr** record to store the paragraph’s attributes. This record offers a simple collection of paragraph justifications, tab stops and margins, and its definition is given in Code Display 10-13 on page ◆ 610. If you need other paragraph styles not found in this record, you should not set VTSF_DEFAULT_PARA_ATTRS.

By default, text objects using the default paragraph attributes will be left-justified, have default tab stops at one inch intervals, and have zero right, left, and paragraph margins.

If you do not use the default paragraph attributes record, you must use the **VisTextParaAttr** structure to store your text object's paragraph attributes. (In this case, do not set VTSF_DEFAULT_PARA_ATTRS in *VTI_storageFlags*.) The text object's paragraph attributes instance field will contain a ChunkHandle to this structure instead of the **VisTextDefaultParaAttr** record.

Code Display 10-18 Setting VisTextParaAttr Attributes

```
/* This example shows setting the initial default paragraph attributes. */
@object GenTextClass MyTextObject = {
    GTXI_text = “”;
    ATTR_GEN_TEXT_DEFAULT_PARA_ATTR = ((J_CENTER << VTDPA_JUSTIFICATION_OFFSET) |
                        ( (2*2) << VTDPA_LEFT_MARGIN_OFFSET) |
                        ( (1*2) << VTDPA_PARA_MARGIN_OFFSET) |
                        ( VTDDT_HALF_INCH << VTDPA_DEFAULT_TABS_OFFSET));
}

/* This example shows setting non-default singular paragraph attributes. */
@object GenTextClass MyTextObject = {
    GTXI_text = “”;
    ATTR_GEN_TEXT_PARA_ATTR = (ChunkHandle) @MyParaAttrs;
}

@chunk VisTextParaAttr MyParaAttrs =
    PARA_ATTR_STYLE_JUST_LEFT_RIGHT_PARA(0, 0, J_CENTER, 0, 2, 1);
```

10.4.4.2 Multiple Paragraph Attributes

To allow your VisText object to exhibit individual paragraph attributes, you should set VTSF_MULTIPLE_PARA_ATTRS and clear the VTSF_DEFAULT_PARA_ATTR flag in the object's *VTI_storageFlags* record. For GenText, set the attribute ATTR_GEN_TEXT_MULTIPLE_PARA_ATTR_RUNS. Any multiple paragraph attribute object will not be able to use any of the default paragraph attributes of type **VisTextDefaultParaAttr**.

Multiple paragraph attributes allow your object's individual paragraphs to exhibit different attributes. One paragraph may be left justified; another may be centered with different margins. Any paragraph may exhibit any of the attributes allowed within the **VisTextParaAttr** structure separately.

Multiple paragraph attributes are stored in the same manner as multiple character attributes. See "Character Runs" on page 638 for a full description of how runs are stored.

Paragraph attributes are specified by runs. Each paragraph run is used until another run is encountered, at which point the new paragraph attributes are applied and used. Runs are defined by the *character* position within a chunk array and an associated token element; therefore, you must take care when defining a paragraph run so that it coincides with the first character of a new paragraph. At the character position, the token corresponds to a **VisTextParaAttr** element. The paragraph attributes specified in this **VisTextParaAttr** structure are used until the next **VisTextParaAttr** element.

10.4.4.3 Changing Paragraph Attributes

MSG_VIS_TEXT_GET_PARA_ATTR,VisTextParaAttrFlags, VisTextParaAttrBorderFlags, VisTextParaAttrDiffs, MSG_VIS_TEXT_SET_PARA_ATTR_BY_DEFAULT, MSG_VIS_TEXT_SET_PARA_ATTR, MSG_VIS_TEXT_SET_PARA_ATTRIBUTES, MSG_VIS_TEXT_ADD_PARA_ATTR

There may be cases in which you would like to change the attributes exhibited by certain paragraphs in your Text object. As with character attributes, you can do this most easily by including one of the controllers mentioned at the end of this chapter.

If you wish to manually change these character attributes, however, there are several messages to retrieve and set character attributes.

■ MSG_VIS_TEXT_GET_PARA_ATTR

```
word    MSG_VIS_TEXT_GET_PARA_ATTR(@stack
        VisTextGetAttrFlags    flags,
        VisTextParaAttrDiffs   *diffs,
        VisTextParaAttr        *attrs,
        dword                  rangeEnd,
        dword                  rangeStart);
```

This message returns a buffer filled in with the **VisTextParaAttr** attributes of the given range of text. If VTGAF_MERGE_WITH_PASSED is passed in the

VisTextGetAttrFlags, then a passed **VisTextParaAttr** structure will be merged with the range of text that this message is sent to. (If this flag is not passed, any information initially in that buffer will be ignored.)

Over the passed range, different paragraphs may exhibit different attributes; a pointer to a **VisTextParaAttrDiffs** structure is also passed to store information about attributes that are different across the range of text. This structure contains lists of **VisTextParaAttrFlags** and **VisTextParaAttrBorderFlags** specifying what paragraph and paragraph border attributes are multiply present. The structure also contains the bitfields of the **VisTextParaAttrAttributes, VisTextHyphenationInfo, VisTextKeepInfo,** and **VisTextDropCapInfo** present over the range of text.

Possible **VisTextParaAttrFlags**. If any of these are set, the selection exhibits the indicated multiple property over the range of text:

VTPAF_MULTIPLE_LEFT_MARGINS
VTPAF_MULTIPLE_RIGHT_MARGINS
VTPAF_MULTIPLE_PARA_MARGINS
VTPAF_MULTIPLE_LINE_SPACINGS
VTPAF_MULTIPLE_DEFAULT_TABS
VTPAF_MULTIPLE_TOP_SPACING
VTPAF_MULTIPLE_BOTTOM_SPACING
VTPAF_MULTIPLE_LEADINGS
VTPAF_MULTIPLE_BG_COLORS
VTPAF_MULTIPLE_BG_GRAY_SCREENS
VTPAF_MULTIPLE_BG_PATTERNS
VTPAF_MULTIPLE_TAB_LISTS
VTPAF_MULTIPLE_STYLES
VTPAF_MULTIPLE_PREPEND_CHARS
VTPAF_MULTIPLE_PARA_NUMBERS

Possible **VisTextParaAttrBorderFlags**:

VTPABF_MULTIPLE_BORDER_LEFT
VTPABF_MULTIPLE_BORDER_TOP
VTPABF_MULTIPLE_BORDER_RIGHT
VTPABF_MULTIPLE_BORDER_BOTTOM
VTPABF_MULTIPLE_BORDER_DOUBLES
VTPABF_MULTIPLE_BORDER_DRAW_INNERS
VTPABF_MULTIPLE_BORDER_ANCHORS
VTPABF_MULTIPLE_BORDER_WIDTHS
VTPABF_MULTIPLE_BORDER_SPACINGS

Objects

VTPABF_MULTIPLE_BORDER_SHADOWS
VTPABF_MULTIPLE_BORDER_COLORS
VTPABF_MULTIPLE_BORDER_GRAY_SCREENS
VTPABF_MULTIPLE_BORDER_PATTERNS

Structures:

```
typedef struct {
      VisTextParaAttrFlags          VTPAD_diffs;
      VisTextParaAttrBorderFlags
VTPAD_borderDiffs;
      VisTextParaAttrAttributes
VTPAD_attributes;
      VisTextHyphenationInfo
VTPAD_hyphenationInfo;
      VisTextKeepInfo               VTPAD_keepInfo;
      VisTextDropCapInfo
VTPAD_dropCapInfo;
} VisTextParaAttrDiffs;
```

Source: Unrestricted.

Destination: Any text object.

Parameters: *flags* VTGAF_MERGE_WITH_PASSED to merge the retrieved text attributes with the text in the passed range.

diffs Pointer to a **VisTextParaAttrDiffs** structure to store attribute differences.

attrs Pointer to a **VisTextParaAttr** buffer to store the retrieved paragraph attributes. If VTGAF_MERGE_WITH_PASSED is passed in flags, this buffer initially contains attributes to match against the retrieved attributes.

rangeEnd End of the range (character position).

rangeStart Beginning of the range (character position).

Return: The token of the specific paragraph attribute run (word value) if the text object is storing runs of **VisTextParaAttr** structures. The *attrs* and *diffs* buffers are also filled with their relevant information.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_SET_PARA_ATTR_BY_DEFAULT

```
void        MSG_VIS_TEXT_SET_PARA_ATTR_BY_DEFAULT(@stack
            VisTextDefaultParaAttr     defParaAttrs,
            dword                      rangeEnd,
            dword                      rangeStart);
```

This message sets the paragraph attributes passed in **VisTextDefaultParaAttr** over the specified range of the text object. If the text object is not in "default" paragraph attribute mode (i.e. it is storing runs of **VisTextParaAttr** structures) it will translate the default attributes into their matching **VisTextParaAttr** attributes.

Source: Unrestricted.

Destination: Any Text object.

Parameters:

defParaAttrs	**VisTextDefaultParaAttr** record to set over the passed range of text.
rangeEnd	End of the range (Character position).
rangeStart	Start of the range (Character position).

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_SET_PARA_ATTR

```
void        MSG_VIS_TEXT_SET_PARA_ATTR(@stack
            VisTextParaAttr     *newParaAttrs,
            dword               rangeEnd,
            dword               rangeStart);
```

This message sets the paragraph attributes passed in the **VisTextParaAttr** buffer over the specified range of the text object. If the text object is in "default" paragraph attribute mode, it will force the text object to begin storing runs of **VisTextParaAttr** structures.

Source: Unrestricted.

Destination: Any Text object.

Parameters:

newParaAttrs	Pointer to a **VisTextParaAttr** buffer.
rangeEnd	End of the range (Character position).
rangeStart	Start of the range (Character position).

Interception: Generally not intercepted.

◆Objects

■ MSG_VIS_TEXT_ADD_PARA_ATTR

```
word      MSG_VIS_TEXT_ADD_PARA_ATTR(
          VisTextMaxParaAttr  *paraAttr);
```

This message adds a given set of paragraph attributes passed in the **VisTextParaAttr** buffer to the paragraph "run" for that text object. The text object must not be in "default" paragraph attribute mode; it must already be storing runs of **VisTextParaAttr** structures.

Source: Unrestricted.

Destination: Any Text object that is storing runs of **VisTextParaAttr** structures.

Parameters: *paraAttr* *P*ointer to a **VisTextParaAttr** buffer.

Return: Token of the paragraph run entry (word-value).

Interception: Generally not intercepted.

10.5 Using VisText

This section describes how to use a VisText object. At the time of documentation printing, **VisTextClass** is undergoing improvement and will therefore be covered fully in future documentation releases. This section currently gives much useful information about the class, however.

Code Display 10-19 shows all the instance data fields of **VisTextClass**. Following the display are descriptions of the individual fields.

Code Display 10-19 VisText Instance Fields

```
/* These are all the instance data fields of the VisText. Many of them are used
 * internally by the class and should not be used by applications. Because they
 * are documented as internal, Geoworks reserves the right to change their meaning
 * or use at any time. Those that are internal are clearly noted as such. */

@instance ChunkHandle          VTI_text;
@instance word                 VTI_charAttrRuns = VIS_TEXT_INITIAL_CHAR_ATTR;
@instance word                 VTI_paraAttrRuns = VIS_TEXT_INITIAL_PARA_ATTR;
@instance VMFileHandle         VTI_vmFile = NullHandle;
@instance word                 VTI_lines = 0;                 /* INTERNAL */
@instance VisTextStorageFlags  VTI_storageFlags = (VTSF_DEFAULT_CHAR_ATTR |
                                     VTSF_DEFAULT_PARA_ATTR);
```

```
@instance VisTextFeatures       VTI_features = 0;
@instance VisTextStates         VTI_state = 0;
@instance VisTextIntFlags       VTI_intFlags = 0;                /* INTERNAL */
@instance VisTextIntSelFlags    VTI_intSelFlags = 0;             /* INTERNAL */
@instance GSRefCountAndFlags    VTI_gsRefCount = 0;              /* INTERNAL */
@instance GStateHandle          VTI_gstate = NullHandle;         /* INTERNAL */
@instance word                  VTI_gstateRegion = -1;           /* INTERNAL */
@instance dword                 VTI_selectStart = 0;
@instance dword                 VTI_selectEnd = 0;
@instance PointDWord            VTI_startEventPos = {0,0};       /* INTERNAL */
@instance dword                 VTI_selectMinStart = 0;          /* INTERNAL */
@instance dword                 VTI_selectMinEnd =0;             /* INTERNAL */
@instance dword                 VTI_lastOffset = 0;              /* INTERNAL */
@instance word                  VTI_goalPosition = 0;            /* INTERNAL */
@instance Point                 VTI_cursorPos = {0,0};           /* INTERNAL */
@instance word                  VTI_cursorRegion = 0;            /* INTERNAL */
@instance word                  VTI_leftOffset = 0;              /* INTERNAL */
@instance byte                  VTI_lrMargin = 0;
@instance byte                  VTI_tbMargin = 0;
@instance ColorQuad             VTI_washColor = {
                                        C_WHITE,
                                        CF_INDEX,
                                        0, 0 }
@instance word                  VTI_maxLength = 10000;
@instance VisTextFilters        VTI_filters = 0;
@instance optr                  VTI_output;
@instance WBFixed               VTI_height = {0,0};              /* INTERNAL */
@instance word                  VTI_lastWidth = -1;              /* INTERNAL */
@instance TimerHandle           VTI_timerHandle = NullHandle;    /* INTERNAL */
@instance word                  VTI_timerID = 0;                 /* INTERNAL */
```

VTI_text stores the ChunkHandle of the object’s text chunk. This chunk will be stored within a chunk in the same resource as the text object. See “The Text” on page 614 for a full discussion of how to manipulate and alter text.

VTI_storageFlags contains flags related to how a VisText object stores its paragraph and character attributes. The composition of these flags affects the function of *VTI_charAttrRuns* and *VTI_paraAttrRuns*.

VTI_charAttrRuns stores the character attributes for the text object. Depending on *VTI_storageFlags*, this instance field may contain a 16-bit record, a ChunkHandle to a **VisTextCharAttr** structure, or a ChunkHandle to a chunk array of character style runs.

VTI_paraAttrRuns stores the paragraph attributes for the text object. Depending on *GTI_storageFlags*, this instance field may contain a 16-bit record, a ChunkHandle to a **VisTextParaAttr** structure, or a ChunkHandle to a chunk array of paragraph style runs.

VTI_features stores a **VisTextFeatures** record to display text within this text object. These features allow such utilities as word wrapping, hyphenation, smart quotes, etc.

VTI_state stores different states of type **VisTextStates** that the text object can operate under, such as editable, selectable, targetable, etc.

VTI_vmFile stores the handle for a text object's associated VM file; this VM file will store the text if the text object incorporates the large model. This instance field is only used for text objects using the large model.

VTI_selectStart stores the character position of the selection's start. *VTI_selectEnd* stores the character position of the selection's end. If both the selection start and selection end coincide, there is no selected text.

VTI_lrMargin stores the left and right margins of the paragraph in points. *VTI_tbMargin* stores the top and bottom margins of the paragraph in points.

VTI_washColor stores the background color (of type **ColorQuad**) of the text object.

VTI_maxLength stores the maximum number of characters allowed in this text object (for non-large objects).

VTI_filters stores the **VisTextFilters** in use by this text object. Filters allow your text object to accept or reject certain sets of characters.

VTI_destination stores the destination for actions taken by this text object.

Code Display 10-20 VisText Variable Data

```
@vardata word   ATTR_VIS_TEXT_TYPE_RUNS;
@vardata word   ATTR_VIS_TEXT_GRAPHICS_RUNS;
@vardata word   ATTR_VIS_TEXT_REGION_ARRAY;
@vardata word   ATTR_VIS_TEXT_STYLE_ARRAY;
@vardata word   ATTR_VIS_TEXT_NAME_ARRAY;
```

```
@vardata word   ATTR_VIS_TEXT_SELECTED_TAB;
@vardata void   ATTR_VIS_TEXT_DO_NOT_INTERACT_WITH_SEARCH_CONTROL;
@vardata VisTextExtendedFilterType ATTR_VIS_TEXT_EXTENDED_FILTER;

typedef struct {
    word                VTSD_count;
    VisTextRange        VTSD_recalcRange;
    VisTextRange        VTSD_selectRange;
    dword               VTSD_showSelectionPos;
    WordFlags           VTSD_notifications;
    byte                VTSD_needsRecalc;
} VisTextSuspendData;

@vardata VisTextSuspendData ATTR_VIS_TEXT_SUSPEND_DATA;

@vardata void   ATTR_VIS_TEXT_NOTIFY_CONTENT;
@vardata void   ATTR_VIS_TEXT_DO_NOT_NOTIFY_CONTENT;
@vardata void   ATTR_VIS_TEXT_SEND_CONTEXT_NOTIFICATIONS_EVEN_IF_NOT_FOCUSED;

typedef struct {
    wchar       VTCFD_startOfRange;
    wchar       VTCFD_endOfRange;
} VisTextCustomFilterData;

@vardata ChunkHandle ATTR_VIS_TEXT_CUSTOM_FILTER;

@vardata word   ATTR_VIS_TEXT_CHAR_ATTR_INSERTION_TOKEN;
@vardata word   ATTR_VIS_TEXT_TYPE_INSERTION_TOKEN;

@vardata void   ATTR_VIS_TEXT_UPDATE_VIA_PROCESS;
@vardata void   ATTR_VIS_TEXT_DOES_NOT_ACCEPT_INK;
@vardata WBFixedATTR_VIS_TEXT_MINIMUM_SINGLE_LINE_HEIGHT;

@vardata word   ATTR_VIS_TEXT_ADD_SPACE_TO_ALL_LINES;
```

ATTR_VIS_TEXT_TYPE_RUNS specifies the Type runs for this text object. You should not need type runs unless you need hyperlink capabilities. ATTR_VIS_TEXT_GRAPHIC_RUNS specifies a graphics run to use within the text object.

ATTR_VIS_TEXT_REGION_ARRAY specifies the array of regions defined for this text object. Text within the VisText will flow from each of the defined regions to the next region. ATTR_VIS_TEXT_STYLE_ARRAY specifies the name array of styles used in the VisText's style sheets.

Objects

ATTR_VIS_TEXT_SELECTED_TAB specifies the position of the currently-selected tab, if any.

ATTR_VIS_TEXT_DO_NOT_INTERACT_WITH_SEARCH_CONTROL instructs the text object not to send the search-control notification.

ATTR_VIS_TEXT_EXTENDED_FILTERS stores the **VisTextExtendedFilterType** to use instead of the **VisTextFilters** stored within *VTI_filters*. Extended filters offer more powerful (and complex) filtering mechanisms for accepting and rejecting text. Including this attribute will affect the performance of your text object, however.

ATTR_VIS_TEXT_SUSPEND_DATA stores data to be used by the text object during a SUSPEND/UNSUSPEND operation (**VisTextSuspendData**).

ATTR_VIS_TEXT_SEND_CONTEXT_NOTIFICATIONS_EVEN_IF_NOT_FOCUSE D sends out context notification even if the object is not focused. Applications must make sure that objects with this attribute do not get modified while another text object has the focus.

ATTR_VIS_TEXT_UPDATE_VIA_PROCESS indicates that status updates must be sent via the process object (usually because there are targetable text objects in multiple threads).

ATTR_VIS_TEXT_DOES_NOT_ACCEPT_INK indicates that the VisText object will not accept ink.

ATTR_VIS_TEXT_MINIMUM_SINGLE_LINE_HEIGHT defines the height of one-line text objects so that custom graphics, etc. can be placed inside the text object.

ATTR_VIS_TEXT_ADD_SPACE_TO_ALL_LINES adds additional spacing above all lines in a text object, and it takes a word argument. This optional attribute is intended to be added to objects that use accent characters that may draw above the top of the text's “font box.” This spacing value differs depending on the system that text is implemented on; only use this attribute if redraws are not drawing your text characters properly. This attribute will also affect any text lines that contain borders, adding additional space between the text and the border.

10.5.1 VisText Features

VTI_features, VisTextFeatures, MSG_VIS_TEXT_SET_FEATURES, MSG_VIS_TEXT_GET_FEATURES

VTI_features stores a bitfield of **VisTextFeatures** specifying whether a particular feature is enabled in the Text object. These flags are listed below.

VTF_NO_WORD_WRAPPING
If set, no word-wrapping is allowed. Characters added after the end of the line will start at the next line.

VTF_AUTO_HYPHENATE
If set, the text object will auto-hyphenate words that span a line break.

VTF_ALLOW_SMART_QUOTES
If set, smart quotes will be allowed if they are enabled.

VTF_ALLOW_UNDO
If set, undo operations will be allowed on the text object. (The text object will handle MSG_META_UNDO.)

VTF_SHOW_HIDDEN_TEXT
If set, hidden text will be displayed. This feature is currently not implemented.

VTF_OUTLINE_MODE
If set, text will be shown in outline mode. This feature is currently not implemented.

VTF_DONT_SHOW_SOFT_PAGE_BREAKS
If set, any soft page breaks (i.e. any page breaks that are not C_PAGE_BREAK) will be ignored.

VTF_DONT_SHOW_GRAPHICS
If set, any graphics in the text object will not be displayed.

VTF_TRANSPARENT
If set, the text object will not do a background wash behind the text after it draws.

VTF_USE_50_PCT_TEXT_MASK
If set, the text object will use a 50% draw mask when drawing, regardless of other character attribute runs. This is used by the GenText object to show a disabled state.

■ MSG_VIS_TEXT_SET_FEATURES

```
void        MSG_VIS_TEXT_SET_FEATURES(
            VisTextFeatures     bitsToSet,
            VisTextFeatures     bitsToClear);
```

This message sets a text object's **VisTextFeatures** (*VTI_features*).

Source: Unrestricted.

Destination: Any text object.

Parameters: *bitsToSet* **VisTextFeatures** to add.

bitsToClear **VisTextFeatures** to remove.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_GET_FEATURES

```
VisTextFeatures MSG_VIS_TEXT_GET_FEATURES();
```

This message retrieves the text object's **VisTextFeatures** (*VTI_features*).

Source: Unrestricted.

Destination: Any text object.

Parameters: None.

Return: **VisTextFeatures** in use by the text object.

Interception: Generally not intercepted.

10.5.2 VisText States

```
VTI_state, VisTextStates, MSG_VIS_TEXT_GET_STATE,
MSG_VIS_TEXT_GET_USER_MODIFIED_STATE,
MSG_VIS_TEXT_SET_NOT_USER_MODIFIED,
MSG_VIS_TEXT_SET_USER_MODIFIED,
MSG_VIS_TEXT_ENTER_OVERSTRIKE_MODE,
MSG_VIS_TEXT_ENTER_INSERT_MODE
```

VTI_state stores a bitfield of **VisTextStates** that determine the ability of the user to interact with the text object. These flags are listed below.

VTS_EDITABLE
If set, the text object is editable. You must set this flag to allow the user to write text into a text object.

VTS_SELECTABLE
If set, text within the text object may be selected. Text may be selectable without it being editable; it can be copied but not cut in that case.

VTS_TARGETABLE
If set, the text object is capable of acting in the target hierarchy. This flag must be set for text objects that interact with controllers.

VTS_ONE_LINE
If set, this object is a simple one-line text object. The text will not scroll vertically, though it may scroll horizontally.

VTS_SUBCLASS_VIRT_PHYS_TRANSLATION
If set, the mapping of virtual attributes to physical attributes will be effected by sending a message (which can be subclassed) rather than through a routine. Normally, this mapping is done through the faster direct routine call, but this flag allows application-specific operations.

VTS_OVERSTRIKE_MODE
If set, the text object is in overstrike mode, not its default insertion mode. Text entered at the cursor will overwrite following text; it will not insert that text at the cursor position.

VTS_USER_MODIFIED
If set, the text inside the object has been changed since the last time the text object sent out its apply message in *GTXI_applyMsg*. You can send the text object MSG_VIS_TEXT_SET_NOT_USER_MODIFIED or MSG_VIS_TEXT_SET_USER_MODIFIED to clear or set this flag.

■ MSG_VIS_TEXT_GET_STATE

```
VisTextStates MSG_VIS_TEXT_GET_STATE();
```

This message retrieves the text object's **VisTextStates** (*VTI_state*).

Source: Unrestricted.

Destination: Any text object.

◆ Objects

Parameters: None.

Return: **VisTextStates** in use by the text object.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_GET_USER_MODIFIED_STATE

```
word        MSG_VIS_TEXT_GET_USER_MODIFIED_STATE();
```

This message returns the state of the text object's VTS_MODIFIED flag in its *VTI_state* entry.

Source: Unrestricted.

Destination: Any text object.

Parameters: None.

Return: *true* **if object has been modified.**

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_SET_NOT_USER_MODIFIED

```
void        MSG_VIS_TEXT_SET_NOT_USER_MODIFIED();
```

This message clears the text object's VTS_USER_MODIFIED flag.

Source: Unrestricted.

Destination: Any text object.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_SET_USER_MODIFIED

```
void        MSG_VIS_TEXT_SET_USER_MODIFIED();
```

This message sets the text object's VTS_USER_MODIFIED flag.

Source: Unrestricted.

Destination: Any text object.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_MODIFY_EDITABLE_SELECTABLE

```
void    MSG_VIS_TEXT_MODIFY_EDITABLE_SELECTABLE(
        VisTextStates       setBits,
        VisTextStates       clearBits);
```

This message modifies the text object's VTS_EDITABLE and VTS_SELECTABLE states. These are the only VisTextStates you should modify with this message.

10.5 **Source:** Unrestricted.

Destination: Any text object.

Parameters: *setBits* The VTS_EDITABLE and/or VTS_SELECTABLE states to set.

clearBits The VTS_EDITABLE and/or VTS_SELECTABLE states to clear.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_ENTER_OVERSTRIKE_MODE

```
void    MSG_VIS_TEXT_ENTER_OVERSTRIKE_MODE(
        Boolean             calledFromTextObject);
```

This message enters the text object into overstrike mode (setting the VTS_OVERSTRIKE_MODE flag).

Source: Unrestricted.

Destination: Any text object.

Parameters: *calledFromTextObject*
true if this message was called internally by the text object.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_ENTER_INSERT_MODE

```
void    MSG_VIS_TEXT_ENTER_INSERT_MODE(
        Boolean             calledFromTextObject);
```

This message enters the text object into insert mode (clearing the VTS_OVERSTRIKE_MODE flag).

Source: Unrestricted.

Destination: Any text object.

Parameters: *calledFromTextObject*
true if this message was called internally by the text object.

Interception: Generally not intercepted.

10.5.3 VisText VM File Storage

```
VTI_vmFile, MSG_VIS_TEXT_SET_VM_FILE,
MSG_VIS_TEXT_SAVE_TO_DB_ITEM,
MSG_VIS_TEXT_SAVE_TO_DB_ITEM_WITH_STYLES,
MSG_VIS_TEXT_LOAD_FROM_DB_ITEM,
MSG_VIS_TEXT_LOAD_FROM_DB_ITEM_WITH_STYLES
```

VTI_vmFile stores the VM file handle that is associated with this text object. This file is the default VM file that will be used for operations that require a storage area. You can set this file handle with MSG_VIS_TEXT_SET_VM_FILE.

To save a text object's current text (including attributes) you can send the text object MSG_VIS_TEXT_SAVE_TO_DB_ITEM. This message takes several **VisTextSaveDBFlags** which specify what aspects of the text to save to the DB item. These flags are listed below:

VTSDBF_TEXT
VTSDBF_CHAR_ATTR
VTSDBF_PARA_ATTR
VTSDBF_TYPE
VTSDBF_GRAPHIC
VTSDBF_STYLE
VTSDBF_REGION
VTSDBF_NAME

To retrieve text saved to a DB item in such a manner, use MSG_VIS_TEXT_LOAD_FROM_DB_ITEM. The text object is able to figure out what features of the text were saved and bring it up with any attributes that were saved previously.

■ MSG_VIS_TEXT_SET_VM_FILE

```
void        MSG_VIS_TEXT_SET_VM_FILE(
            VMFileHandle        file);
```

This message sets the text object's *VTI_vmFile* instance field to the passed VM file handle. This file handle will be used as the default for operations that require a VM file.

Source: Unrestricted.

Destination: Any text object.

Parameters: *file* VM file handle.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_SAVE_TO_DB_ITEM

```
DBGroupAndItem MSG_VIS_TEXT_SAVE_TO_DB_ITEM(
            DBGroupAndItem      item,
            VisTextSaveDBFlags  flags);
```

This message saves information about the current state of the text object. Data saved with this message can be retrieved with MSG_VIS_TEXT_LOAD_FROM_DB_ITEM.

Source: Unrestricted.

Destination: Any text object.

Parameters: *item* DB group and item to store the text information. If this is left zero, the info will be saved to the VM file specified in *VTI_vmFile*.

flags **VisTextSaveDBFlags** specifying what aspect of the text's info to save.

Return: **DBItem** of saved text info.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_SAVE_TO_DB_ITEM_WITH_STYLES

```
DBGroupAndItem MSG_VIS_TEXT_SAVE_TO_DB_ITEM_WITH_STYLES(
          FileHandle          xferFile,
          VisTextSaveDBFlags  flags,
          DBGroupAndItem      item,
          StyleSheetParams    *params);
```

This message saves information about the current state of the text object, passing a **StyleSheetParams** structure.

VTST_RUNS_ONLY should be passed in the **VisTextSaveDBFlags** entries VTSDBF_CHAR_ATTR and VTSDBF_PARA_ATTR. Do not set VTSDBF_STYLE.

Source: Unrestricted.

Destination: Any text object.

Parameters: *xferFile* If non-zero, the **StyleSheetParams** structure is not initialized and this is the file to put the style sheet arrays into.

flags **VisTextSaveDBFlags** specifying what aspect of the text's info to save.

item DB group and item to store the text information. If this is left zero, the info will be saved to the VM file specified in *VTI_vmFile*.

params* Pointer to a **StyleSheetParams structure.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_LOAD_FROM_DB_ITEM

```
void      MSG_VIS_TEXT_LOAD_FROM_DB_ITEM(
          DBGroupAndItem      item,
          VMFileHandle        file);
```

This message loads text saved with MSG_VIS_TEXT_SAVE_TO_DB_ITEM.

Source: Unrestricted.

Destination: Any text object.

Parameters: *item* **DBItem** to load.

file Handle of the VM file to use. If zero, the VM file in *VTI_vmFile* will be used.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_LOAD_FROM_DB_ITEM_WITH_STYLES

```
void      MSG_VIS_TEXT_LOAD_FROM_DB_ITEM_WITH_STYLES(
          FileHandle          file,
          DBGroupAndItem      item,
          StyleSheetParams    *params);
```

This message loads text and style sheets saved with MSG_VIS_TEXT_SAVE_TO_DB_ITEM_WITH_STYLES.

Source: Unrestricted.

Destination: Any text object.

Parameters: *file* Handle of the VM file to use. If zero, the VM file in *VTI_vmFile* will be used.

item **DBItem** to load.

params* Pointer to a **StyleSheetParams structure.

Interception: Generally not intercepted.

10.5.4 Text Filters

```
VTI_filters, VisTextFilters, VisTextFilterClass,
ATTR_VIS_TEXT_EXTENDED_FILTER, MSG_VIS_TEXT_SET_FILTER,
MSG_VIS_TEXT_GET_FILTER
```

When the user enters text into the text object, the text object is able to filter out characters that it does not wish to be displayed. This behavior is produced through use of **VisTextFilters**.

VTI_filters stores a **VisTextFilters** record. This record has several entries:

VTF_NO_SPACES filters out any space characters.

VTF_NO_TABS filters out any tab characters.

VTF_UPCASE_CHARS ensures that any alphabetic characters are made uppercase. Other characters are unaffected.

VTF_FILTER_CLASS stores a **VisTextFilterClass**. This value can be any one of the following:

VTFC_NO_FILTER
VTFC_ALPHA
VTFC_NUMERIC
VTFC_SIGNED_NUMERIC
VTFC_SIGNED_DECIMAL
VTFC_FLOAT_DECIMAL
VTFC_ALPHA_NUMERIC
VTFC_FILENAMES
VTFC_DOS_FILENAMES
VTFC_DOS_PATH
VTFC_DATE
VTFC_TIME
VTFC_DASHED_ALPHA_NUMERIC
VTFC_NORMAL_ASCII
VTFC_DOS_VOLUME_NAMES
VTFC_DOS_CHARACTER_SET
VTFC_ALLOW_COLUMN_BREAKS

For more complex filtering, you can include ATTR_VIS_TEXT_EXTENDED_FILTER in your text object. Each value corresponds to a different message which you can intercept to provide custom filtering. These values are:

VTEFT_REPLACE_PARAMS
This filter generates MSG_VIS_TEXT_FILTER_VIA_REPLACE_PARAMS. This message passes a **VisTextReplaceParameters** structure that you can either accept or reject as a whole. This will be supported in an upcoming release.

VTEFT_CHARACTER_LEVELER_LEVEL
This filter generates MSG_VIS_TEXT_FILTER_VIA_CHARACTER. This message passes a single character that you can either accept or reject.

VTEFT_BEFORE_AFTER
This filter generates MSG_VIS_TEXT_FILTER_VIA_BEFORE_AFTER. This message passes two buffers containing the current text and the proposed new text. This will be supported in an upcoming release.

■ MSG_VIS_TEXT_GET_FILTER

```
byte    MSG_VIS_TEXT_GET_FILTER();
```

This message retrieves the current **VisTextFilters** in use by the text object (stored in *VTI_filters*).

Source: Unrestricted.

Destination: Any text object.

10.5 **Return:** **VisTextFilters** in use by the text object.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_SET_FILTER

```
void    MSG_VIS_TEXT_SET_FILTER(
        byte            filter);
```

This message sets the text object's **VisTextFilters** (stored in *VTI_filters*) to the passed value. Resetting this filter does not affect any text that already exists in the text object.

Source: Unrestricted.

Destination: Any text object.

Parameters: *filter* **VisTextFilters** to set.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_FILTER_VIA_CHARACTER

```
word    MSG_VIS_TEXT_FILTER_VIA_CHARACTER(
        word            charToFilter);
```

This message allows the text object to filter text on a character by character basis. This message is called for each character that is entered into the text object.

This message will be generated if the text object includes ATTR_VIS_TEXT_EXTENDED_FILTER in its instance data, with its value set to the **VisTextExtendedFilterType** VTEFT_CHARACTER_LEVELER_LEVEL.

Source: Sent by the system if text object contains ATTR_VIS_TEXT_EXTENDED_FILTER set to VTEFT_CHARACTER_LEVELER_LEVEL.

Destination: Any text object

Parameters: *charToFilter* Character value that is being filtered. You can pass this value on to the superclass or filter it out.

Interception: Must intercept to provide character by character filtering.

10.5.5 Key Functions

`VisTextKeyFunction, MSG_VIS_TEXT_DO_KEY_FUNCTION`

The VisText object is able to recognize certain generic functions that operate on text. Most of these functions are initiated through a key sequence entered by the user, but you can mimic such a key sequence by sending the text object MSG_VIS_TEXT_DO_KEY_FUNCTION, passing it a **VisTextKeyFunction**.

The following **VisTextKeyFunction** values are recognized by the text object:

VTKF_FORWARD_LINE
VTKF_BACKWARD_LINE
VTKF_SELECT_ADJUST_FORWARD_LINE
VTKF_SELECT_ADJUST_BACKWARD_LINE
VTKF_FORWARD_CHAR
VTKF_BACKWARD_CHAR
VTKF_FORWARD_WORD
VTKF_BACKWARD_WORD
VTKF_FORWARD_PARAGRAPH
VTKF_BACKWARD_PARAGRAPH
VTKF_START_OF_LINE
VTKF_END_OF_LINE
VTKF_START_OF_TEXT
VTKF_END_OF_TEXT
VTKF_SELECT_WORD
VTKF_SELECT_LINE
VTKF_SELECT_PARAGRAPH
VTKF_SELECT_OBJECT
VTKF_SELECT_ADJUST_FORWARD_CHAR
VTKF_SELECT_ADJUST_BACKWARD_CHAR
VTKF_SELECT_ADJUST_FORWARD_WORD
VTKF_SELECT_ADJUST_BACKWARD_WORD
VTKF_SELECT_ADJUST_FORWARD_PARAGRAPH
VTKF_SELECT_ADJUST_BACKWARD_PARAGRAPH

VTKF_SELECT_ADJUST_TO_START
VTKF_SELECT_ADJUST_TO_END
VTKF_SELECT_ADJUST_START_OF_LINE
VTKF_SELECT_ADJUST_END_OF_LINE
VTKF_DELETE_BACKWARD_CHAR
VTKF_DELETE_BACKWARD_WORD
VTKF_DELETE_BACKWARD_LINE
VTKF_DELETE_BACKWARD_PARAGRAPH
VTKF_DELETE_TO_START
VTKF_DELETE_CHAR
VTKF_DELETE_WORD
VTKF_DELETE_LINE
VTKF_DELETE_PARAGRAPH
VTKF_DELETE_TO_END
VTKF_DELETE_EVERYTHING
VTKF_DESELECT
VTKF_TOGGLE_OVERSTRIKE_MODE
VTKF_TOGGLE_SMART_QUOTES

■ MSG_VIS_TEXT_DO_KEY_FUNCTION

```
void      MSG_VIS_TEXT_DO_KEY_FUNCTION
          VisTextKeyFunction     func,
          word                   data);
```

This message allows the text object to mimic certain text operations that normally are the result of user keystrokes.

Note: the **VisTextKeyFunction** types VTKF_SELECT_LINE, VTKF_SELECT_PARAGRAPH, and VTKF_SELECT_OBJECT will be supported in an upcoming release.

Source: Unrestricted.

Destination: Any text object

Parameters: *func* **VisTextKeyFunction** to perform.

data Data for that key function (if needed).

Interception: Generally not intercepted.

10.5.6 Setting Text Confines

```
MSG_VIS_TEXT_GET_MAX_LENGTH, MSG_VIS_TEXT_SET_MAX_LENGTH,
MSG_VIS_TEXT_GET_LR_MARGIN, MSG_VIS_TEXT_SET_LR_MARGIN,
MSG_VIS_TEXT_GET_TB_MARGIN, MSG_VIS_TEXT_SET_TB_MARGIN
```

■ MSG_VIS_TEXT_GET_MAX_LENGTH

```
word        MSG_VIS_TEXT_GET_MAX_LENGTH();
```

This message returns the maximum length of text in the text object (stored in *VTI_maxLength*).

Source: Unrestricted.

Destination: Any text object.

Return: Maximum length of the text, in characters.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_SET_MAX_LENGTH

```
void        MSG_VIS_TEXT_SET_MAX_LENGTH(
            word                 newMaxLength);
```

This message sets the maximum length of text within a text object (stored in *VTI_maxLength*). If the current text is too long, any excess text is removed.

Source: Unrestricted.

Destination: Any text object (except **VisLargeTextClass**).

Parameters: *newMaxLength* Maximum length of the text object (in characters).

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_GET_LR_MARGIN

```
byte        MSG_VIS_TEXT_GET_LR_MARGIN();
```

This message returns the current left/right margins. Both margins are the same and are thus represented by the same returned value.

Source: Unrestricted.

Destination: **VisTextClass** objects (not GenText objects).

Return: The current left/right margin.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_SET_LR_MARGIN

```
void     MSG_VIS_TEXT_SET_LR_MARGIN(
         byte                lrMargin);
```

This message sets the left/right margins of the text object. Both margins will be set to the same value. Do not use this message on GenText objects, as the specific UI is responsible for this behavior.

Source: Unrestricted.

Destination: **VisTextClass** objects (not GenText objects).

Parameters: *lrMargin* New left/right margins.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_GET_LR_MARGIN

```
byte     MSG_VIS_TEXT_GET_LR_MARGIN();
```

This message returns the current left/right margins. Both margins are the same and are thus represented by the same returned value.

Source: Unrestricted.

Destination: **VisTextClass** objects (not GenText objects).

Return: The current left/right margin.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_SET_TB_MARGIN

```
void     MSG_VIS_TEXT_SET_TB_MARGIN(
         byte                tbMargin);
```

This message sets the top/bottom margins of the text object. Both margins will be set to the same value. Do not use this message on GenText objects, as the specific UI is responsible for this behavior.

Source: Unrestricted.

Destination: **VisTextClass** objects (not GenText objects).

Parameters: *tbMargin* New top/bottom margins.

Interception: Generally not intercepted.

10.5.7 Output Messages

```
VTI_output, MSG_VIS_TEXT_GET_OUTPUT,
MSG_VIS_TEXT_SET_OUTPUT, MSG_META_TEXT_USER_MODIFIED,
MSG_META_TEXT_NOT_USER_MODIFIED,MSG_META_TEXT_CR_FILTERED,
MSG_META_TEXT_TAB_FILTERED, MSG_META_TEXT_LOST_FOCUS,
MSG_META_TEXT_GAINED_FOCUS, MSG_META_TEXT_LOST_TARGET,
MSG_META_TEXT_GAINED_TARGET,
MSG_META_TEXT_EMPTY_STATUS_CHANGED
```

VTI_output stores the object to receive messages sent out by the text object. A range of messages imported from **MetaClass** (**MetaTextMessages**) are sent to this destination optr. These messages are listed below:

MSG_META_TEXT_USER_MODIFIED is sent to *VTI_output* when the user first modifies text within the object. This message is also sent to the text object itself to set its VTS_MODIFIED state (after it is sent to *VTI_output*). If the text object's VTS_MODIFIED bit is later cleared, it is sent MSG_META_TEXT_NOT_USER_MODIFIED (and may receive MSG_META_TEXT_USER_MODIFIED again).

MSG_META_TEXT_CR_FILTERED is sent when a Carriage Return is filtered out of the text input by the text object's filtering mechanism. This message is first sent to the text object itself.

MSG_META_TEXT_TAB_FILTERED is sent when a Tab is filtered out of the text input by the text object's filtering mechanism. This message is first sent to the text object itself.

MSG_META_TEXT_LOST_FOCUS is sent when the text object loses the focus. MSG_META_TEXT_GAINED_FOCUS is sent when the text object gains the focus.

MSG_META_TEXT_LOST_TARGET is sent when the text object loses the target. MSG_META_TEXT_GAINED_TARGET is sent when the text object gains the target.

MSG_META_TEXT_EMPTY_STATUS_CHANGED is sent when the text object either changes from being empty to being non-empty or vice-versa.

To retrieve the current output of the text object, send it MSG_VIS_TEXT_GET_OUTPUT. To change the current output, send it MSG_VIS_TEXT_SET_OUTPUT.

■ MSG_META_TEXT_USER_MODIFIED

```
@importMessage MetaTextMessages, void MSG_META_TEXT_USER_MODIFIED(
        optr                obj);
```

MSG_META_TEXT_USER_MODIFIED is sent to *VTI_output* when the user modifies text within the object. The message is also sent to the text object itself (to mark itself modified) after it is sent to the destination, so the state of the text's VTS_MODIFIED bit may not accurately reflect the state of the text object. If you need to intercept this message and also need to perform an operation using the text object's modified state.

Source: Sent by the system when the user modifies the text object.

Destination: *VTI_output* of the text object's instance data. The message is also sent to the text object itself.

Parameters: *obj* Optr of the text object modified.

Interception: May intercept to receive notification of user modification of text. Because modification of the text object's VTS_MODIFIED bit may be asynchronous, you should send any operations that depend on this information (such as clearing the VTS_MODIFIED bit) to the text object using MF_FORCE_QUEUE

■ MSG_META_TEXT_NOT_USER_MODIFIED

```
@importMessage MetaTextMessages, void MSG_META_TEXT_NOT_USER_MODIFIED(
        optr                obj);
```

This message is sent to *VTI_output* when the VTS_MODIFIED state of the text object is cleared, either by the system, or by an application inspired action.

Source: Sent by the system when the user modifies the text object.

Destination: *VTI_output* of the text object's instance data. The message is also sent to the text object itself first.

Parameters: *obj* Optr of the text object marked not modified.

Interception: Intercept to receive notification of when a text object is marked not modified.

■ MSG_META_TEXT_CR_FILTERED

```
@importMessage MetaTextMessages, void MSG_META_TEXT_CR_FILTERED(
        word                character,
        word                flags,
        word                state);
```

This message is sent to *VTI_output* when a Carriage Return is filtered out in the text input stream.

Source: Sent by the system when a CR is filtered out of the text object.

Destination: *VTI_output* of the text object's instance data. The message is also sent to the text object itself.

Parameters: *character* The character value (CR).

flags **ShiftState** and **CharFlags**.

state The state of the text object (*VTI_state*) at the time of the filtering action.

Interception: Intercept to receive notification that a Carriage Return has been filtered out of the text object.

■ MSG_META_TEXT_TAB_FILTERED

```
@importMessage MetaTextMessages, void MSG_META_TEXT_TAB_FILTERED(
        word                character,
        word                flags,
        word                state);
```

This message is sent to *VTI_output* when a Tab is filtered out in the text input stream.

Source: Sent by the system when the user modifies the text object.

Destination: *VTI_output* of the text object's instance data. The message is also sent to the text object itself.

Parameters: *character* The character value (Tab).

flags **ShiftState** and **CharFlags**.

state The state of the text object (*VTI_state*) at the time of the filtering action.

Interception: Intercept to receive notification that a Tab has been filtered out of the text object.

■ MSG_META_TEXT_LOST_FOCUS

```
@importMessage MetaTextMessages, void MSG_META_TEXT_LOST_FOCUS(
        optr                obj);
```

This message is sent when the text object loses the focus of the application.

Source: Sent by the system when the user modifies the text object.

Destination: *VTI_output* of the text object's instance data.

Parameters: *obj* Optr of the text object.

Interception: Intercept to receive notification of when the text object loses the focus.

■ MSG_META_TEXT_GAINED_FOCUS

```
@importMessage MetaTextMessages, void MSG_META_TEXT_GAINED_FOCUS(
        optr                obj);
```

This message is sent when the text object gains the focus of the application.

Source: Sent by the system when the user modifies the text object.

Destination: *VTI_output* of the text object's instance data.

Parameters: *obj* Optr of the text object.

Interception: Intercept to receive notification of when the text object gains the focus.

■ MSG_META_TEXT_LOST_TARGET

```
@importMessage MetaTextMessages, void MSG_META_TEXT_LOST_TARGET(
        optr                obj);
```

This message is sent when the text object loses the target of an application.

Source: Sent by the system when the user modifies the text object.

Destination: *VTI_output* of the text object's instance data.

Parameters: *obj* Optr of the text object.

Interception: Intercept to receive notification of when the text object loses the target.

■ MSG_META_TEXT_GAINED_TARGET

```
@importMessage MetaTextMessages, void MSG_META_TEXT_GAINED_TARGET(
        optr                obj);
```

This message is sent when the text object gains the target of an application.

Source: Sent by the system when the user modifies the text object.

Destination: *VTI_output* of the text object's instance data.

Parameters: *obj* Optr of the text object.

Interception: Intercept to receive notification of when the text object gains the target.

■ MSG_META_TEXT_EMPTY_STATUS_CHANGED

```
@importMessage MetaTextMessages, void
                MSG_META_TEXT_EMPTY_STATUS_CHANGED(
        optr                object,
        Boolean             hasTextFlag);
```

This message is sent out when the text object is either becoming empty or not empty.

Source: Sent by the system when the user modifies the text object.

Destination: *VTI_output* of the text object's instance data. The message is also sent to the text object itself first.

Parameters: *object* Optr of the text object whose empty status is changing.

hasTextFlag Non-zero if the text object is becoming non-empty.

Interception: Intercept to receive notification when the empty status of the text object is changing.

■ MSG_VIS_TEXT_GET_OUTPUT

```
optr    MSG_VIS_TEXT_GET_OUTPUT();
```

This message returns the current destination (output) for actions taken by the text object.

Source: Unrestricted.

Destination: Any text object.

Return: Optr of the object's destination (in the *VTI_output* field).

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_SET_OUTPUT

```
void    MSG_VIS_TEXT_SET_OUTPUT(
        optr                newOutput);
```

This message sets the destination object for actions taken by this text object.

Source: Unrestricted.

Destination: Any text object.

Parameters: *newOutput* New destination object (placed in object's *VTI_output* field).

Interception: Generally not intercepted.

10.5.8 Getting Geometry Information

MSG_VIS_TEXT_GET_MIN_WIDTH,
MSG_VIS_TEXT_GET_AVERAGE_CHAR_WIDTH,
MSG_VIS_TEXT_GET_LINE_HEIGHT,
MSG_VIS_TEXT_RECALC_AND_DRAW,
MSG_VIS_TEXT_GET_ONE_LINE_WIDTH,
MSG_VIS_TEXT_GET_SIMPLE_MIN_WIDTH

■ MSG_VIS_TEXT_GET_MIN_WIDTH

```
word    MSG_VIS_TEXT_GET_MIN_WIDTH();
```

This message returns the minimum width that can be supported for attributes of the text object.

Source: Unrestricted.

Destination: Any text object.

Return: Minimum width (in points) that can be supported.

■ MSG_VIS_TEXT_GET_AVERAGE_CHAR_WIDTH

```
word    MSG_VIS_TEXT_GET_AVERAGE_CHAR_WIDTH();
```

This message returns the average character width for the first font/character attribute run.

Source: Unrestricted.,

Destination: Any text object.

Return: Average character width (in points).

■ MSG_VIS_TEXT_GET_LINE_HEIGHT

```
word    MSG_VIS_TEXT_GET_LINE_HEIGHT();
```

This message returns the height of a line (in points) for a single-line text object *only*.

Source: Unrestricted.

Destination: Single-line text objects that are not within document objects.

Return: Height of line (in points).

■ MSG_VIS_TEXT_RECALC_AND_DRAW

```
void        MSG_VIS_TEXT_RECALC_AND_DRAW();
```

This message recalculates and redraws a text object, usually after constraints on either its width or its height is made without its knowledge.

Source: Unrestricted.

Destination: Any text object.

Interception: Generally not intercepted.

■ MSG_VIS_TEXT_GET_ONE_LINE_WIDTH

```
word        MSG_VIS_TEXT_GET_ONE_LINE_WIDTH(
            word                charsToCalc);
```

This message calculates the width of a single-line text object. This width is calculated using the first encountered character attributes.

Source: Unrestricted.

Destination: Any single-line text object.

Parameters: *charsToCalc* Number of characters (from start of text) to use in the calculation.

Return: Width of the text.

■ MSG_VIS_TEXT_GET_SIMPLE_MIN_WIDTH

```
word        MSG_VIS_TEXT_GET_SIMPLE_MIN_WIDTH();
```

This message returns the minimum width that can be supported for the text object's current text, font and character attributes. This message assumes that the caller is not worried about keeping a width for the largest border possible.

Source: Unrestricted.

Destination: Any text object.

Return: Minimum width which can be supported.

10.6 Using GenText

10.6

GenTextClass provides your application with a generic UI object to both display and edit text. The GenText object is a highly functional object, capable of displaying text to the user under a variety of constraints. The object's role is primarily to display text pertaining to user interface operations. The object also provides a wide array of text processing features, however, to display the text in different formats. For more powerful and diverse text formatting (such as word-processing), you should probably use a VisText object instead.

The GenText has many advanced features of the Text Object Library built in. Most of these features may not be useful for cases where you will most likely use a GenText object; GenText is designed for simpler text edit capabilities than incorporating graphics into text, for example. Still, the GenText object provides a great deal of default behavior and, with a little work, a highly powerful and specifically tailored text object.

The GenText object may not only show text to the user but may also allow the user to supply or edit the displayed text. For example, a file copy operation may request the new name to write the file to; a GenText object would allow the user to enter this file name within its text field.

The text object library offers the following key features:

- Display-only and editable text fields.
- Automatic keyboard support, allowing the user to immediately begin entering text into the GenText's text field.
- Automatic word wrapping from one line to the next (or the capability to disable this option).
- Vertical scrolling of text (providing scrollbars either initially or when needed).
- Selection of text, usually represented by highlighting.
- The specification of actions to perform when the user completes text edits.
- Text character styles to display the text. Sample character styles are bold, underline, strike-through, etc.

- Paragraph styles (rulers) to display the text. Sample paragraph styles are right-justified, paragraph spacing, hidden text, etc.

- Fonts to display the text.

- Ability to store the text in a variety of data structures.

- Interaction with the text controllers.

GenText inherits most of its capabilities from **VisTextClass** which in turn uses the text object library. You may wish to peruse the structure of this library if you want to know more details of how the GenText performs its default behavior.

GenText is used primarily in three situations: First, when you need to display text (such as in a "help" window), the GenText object can act as a non-editable text field with its own scrolling and geometry management. Second, when you need to get textual input not related to word processing or text editing (such as a file name), the GenText can act as an editable field of one or more lines. Third, when simple word processing or text editing in the generic tree is required (as in the NotePad application), the GenText can provide a scrollable, sizable text-edit field. If you want, the editable GenText can even support advanced paragraph and text formatting.

VisText is used in most situations where complex or advanced formatting or publishing features are desired. Typically, a GenView and a VisContent will be used to provide the root of the visible tree and its interaction with the UI. The VisText is much more flexible than the GenText and allows complex publishing and page layout features that would be difficult with GenText. VisText also allows you to have the same interface no matter what specific UI is in use since it is a visible object and not subject to the constraints of the specific UI.

10.6.1 GenText Instance Data

The GenText object has several instance fields which affect the visual presentation of text. These are listed in Code Display 10-21.

10.6

Code Display 10-21 GenText Instance Data

```
@instance ChunkHandle           GTXI_text;
@instance word                  GTXI_maxLength = 32767;
@instance GenTextAttrs          GTXI_attrs = GTA_USE_TAB_FOR_NAVIGATION;
@instance GenTextStateFlags     GTXI_stateFlags = 0;
@instance optr                  GTXI_destination;
@instance word                  GTXI_applyMsg = 0;

/* GenTextAttrs */

typedef ByteFlags GenTextAttrs;
#define GTA_SINGLE_LINE_TEXT             0x80
#define GTA_USE_TAB_FOR_NAVIGATION       0x40
#define GTA_INIT_SCROLLING               0x20
#define GTA_NO_WORD_WRAPPING             0x10
#define GTA_ALLOW_TEXT_OFF_END           0x08
#define GTA_TAIL_ORIENTED                0x04
#define GTA_DONT_SCROLL_TO_CHANGES       0X02

/* GenTextStateFlags */

typedef ByteFlags GenTextStateFlags
#define GTSF_INDETERMINATE               0x80
#define GTSF_MODIFIED                    0x40
```

GTXI_text stores the ChunkHandle of the chunk containing the GenText's text. This chunk will be stored in the same resource as the text object. You may alter the text within this chunk with any of the previously mentioned **VisTextClass** messages. See "The Text" on page 614 for a full discussion of how to manipulate and alter text.

GTXI_maxLength specifies the maximum number of character positions allowed within the GenText. By default, up to 32767 characters may appear within a GenText. If you set *GTXI_maxLength* to a lower number, any characters beyond the maximum length will be ignored.

GTXI_attrs specifies the **GenTextAttrs** to set for the GenText object. These attributes affect the presentation of the text.

GTXI_stateFlags specifies the **GenTextStateFlags** to use with this text object. These state flags affect the modified and indeterminate states of the object.

GTXI_applyMsg sets the message for this object to send out whenever activated. In most cases, your handler for this message may then retrieve the text and perform whatever required operations on that text.

GTXI_destination sets the object or process to handle apply messages sent out by this object.

Code Display 10-22 GenText VarData Instance Fields

```
@vardata Message          ATTR_GEN_TEXT_STATUS_MSG;
@vardata void             ATTR_GEN_TEXT_SET_MODIFIED_ON_REDUNDANT_SELECTION;
@vardata void             ATTR_GEN_TEXT_SELECTABLE;
@vardata VisTextDefaultCharAttr ATTR_GEN_TEXT_DEFAULT_CHAR_ATTR;
@vardata VisTextDefaultParaAttr ATTR_GEN_TEXT_DEFAULT_PARA_ATTR;
@vardata ChunkHandle      ATTR_GEN_TEXT_CHAR_ATTR;
@vardata ChunkHandle      ATTR_GEN_TEXT_MULTIPLE_CHAR_ATTR_RUNS;
@vardata ChunkHandle      ATTR_GEN_TEXT_PARA_ATTR;
@vardata ChunkHandle      ATTR_GEN_TEXT_MULTIPLE_PARA_ATTR_RUNS;

@vardata word             ATTR_GEN_TEXT_EXTENDED_FILTER;

@vardata word             ATTR_GEN_TEXT_TYPE_RUNS;
@vardata word             ATTR_GEN_TEXT_GRAPHIC_RUNS;
@vardata word             ATTR_GEN_TEXT_REGION_ARRAY;
@vardata word             ATTR_GEN_TEXT_STYLE_ARRAY;
@vardata word             ATTR_GEN_TEXT_NAME_ARRAY;

@vardata optr             ATTR_GEN_TEXT_RUNS_ITEM_GROUP;
    @reloc       ATTR_GEN_TEXT_RUNS_ITEM_GROUP;

@vardata void             ATTR_GEN_TEXT_DO_NOT_INTERACT_WITH_SEARCH_CONTROL;
@vardata optr             ATTR_GEN_TEXT_SET_OBJECT_ENABLED_WHEN_TEXT_EXISTS;
@vardata void             ATTR_GEN_TEXT_LEGAL_DOS_PATH;
@vardata void             ATTR_GEN_TEXT_DATE;
@vardata void             ATTR_GEN_TEXT_TIME;
@vardata void             ATTR_GEN_TEXT_MAKE_UPPERCASE;
@vardata void             ATTR_GEN_TEXT_ALLOW_COLUMN_BREAKS;
@vardata void             ATTR_GEN_TEXT_UPPERCASE_ALPHA;
@vardata void             ATTR_GEN_TEXT_UPPERCASE_ALPHA_NUMERIC;
@vardata void             ATTR_GEN_TEXT_DASHED_ALPHA_NUMERIC;
@vardata void             ATTR_GEN_TEXT_NORMAL_ASCII;
@vardata void             ATTR_GEN_TEXT_LEGAL_DOS_VOLUME_NAMES;
@vardata void             ATTR_GEN_TEXT_DOS_CHARACTER_SET;
@vardata void             ATTR_GEN_TEXT_NO_SPACES;
@vardata void             ATTR_GEN_TEXT_ALLOW_SPACES;
@vardata void             ATTR_GEN_TEXT_ALPHA;
```

```
@vardata void           ATTR_GEN_TEXT_NUMERIC;
@vardata void           ATTR_GEN_TEXT_SIGNED_NUMERIC;
@vardata void           ATTR_GEN_TEXT_SIGNED_DECIMAL;
@vardata void           ATTR_GEN_TEXT_FLOAT_DECIMAL;
@vardata void           ATTR_GEN_TEXT_ALPHA_NUMERIC;
@vardata void           ATTR_GEN_TEXT_LEGAL_FILENAMES;
@vardata void           ATTR_GEN_TEXT_LEGAL_DOS_FILENAMES;
@vardata void           ATTR_GEN_TEXT_NEVER_MAKE_SCROLLABLE;
@vardata void           ATTR_GEN_TEXT_DOES_NOT_ACCEPT_INK;
@vardata ChunkHandle    ATTR_GEN_TEXT_CUSTOM_FILTER;
```

ATTR_GEN_TEXT_STATUS_MSG sets a status message for this GenText. A status message allows the GenText to send notification other than its apply message.

ATTR_GEN_TEXT_SET_MODIFIED_ON_REDUNDANT_SELECTION sets the text dirty (modified) on any modification, even if it does not change the text or the text's attributes.

ATTR_GEN_TEXT_SELECTABLE should be set if text should be selectable whether or not the text is editable. By default non-editable text is not selectable. Editable text is always selectable.

ATTR_GEN_TEXT_DEFAULT_CHAR_ATTR stores singular default character attributes of type **VisTextDefaultCharAttr** in a word-length record. ATTR_GEN_TEXT_DEFAULT_PARA_ATTR stores singular default paragraph attributes of type **VisTextDefaultParaAttr** in a word-length record.

ATTR_GEN_TEXT_CHAR_ATTR stores the singular character attributes for the text object. This vardata field stores the ChunkHandle to a single **VisTextCharAttr** structure.

ATTR_GEN_TEXT_MULTIPLE_CHAR_ATTR_RUNS stores the multiple character attributes for the text object. This vardata field will store the ChunkHandle to a chunk array of character style runs (each of type **VisTextCharAttr**). See “Character Runs” on page 638.

ATTR_GEN_TEXT_PARA_ATTR stores the singular paragraph attributes for the text object. This vardata field stores the ChunkHandle to a single **VisTextParaAttr** structure.

ATTR_GEN_TEXT_MULTIPLE_PARA_ATTR_RUNS stores the multiple paragraph attributes for the text object. This vardata field will store the ChunkHandle to a chunk array of paragraph style runs (each of type **VisTextParaAttr**). See “Paragraph Runs” on page 646.

ATTR_GEN_TEXT_EXTENDED_FILTER stores the **VisTextExtendedFilterTypes** to use instead of the default **VisTextFilters**. Extended filters offer more powerful (and complex) filtering mechanisms for accepting and rejecting text into the GenText.

ATTR_GEN_TEXT_TYPE_RUNS specifies the type runs for this text object. You should not need type runs unless you need hyperlink capabilities. ATTR_GEN_TEXT_GRAPHIC_RUNS specifies a graphics run to use within the text object.

ATTR_GEN_TEXT_REGION_ARRAY specifies the array of regions defined for this text object. Text within the GenText will flow from each of the defined regions to the next region. ATTR_GEN_TEXT_STYLE_ARRAY specifies the name array of styles used in the GenText’s style sheets.

ATTR_GEN_TEXT_RUNS_ITEM_GROUP indicates that this text object is linked to an item group (list). When the user hits Return (or the equivalent), the text object sends a message to the item group; similarly, the item group will attempt to update the text object on changes.

ATTR_GEN_TEXT_DO_NOT_INTERACT_WITH_SEARCH_CONTROL indicates that the text object should not send notification to the search control object.

ATTR_GEN_TEXT_SET_OBJECT_ENABLED_WHEN_TEXT_EXISTS makes the specified object enabled any time the GenText object has text in it.

ATTR_GEN_TEXT_LEGAL_DOS_PATH makes sure that only legal DOS paths are used.

ATTR_GEN_TEXT_DATE and ATTR_GEN_TEXT_TIME allows short date and time characters and spaces; the text object will be affected by the localization driver.

ATTR_GEN_TEXT_MAKE_UPPERCASE indicates that in addition to any other filters, any alphabetic characters are forced into uppercase.

ATTR_GEN_TEXT_ALLOW_COLUMN_BREAKS allows all characters including column breaks. Column breaks are normally filtered out in a GenText object.

The column breaks usually will appear as new lines (like carriage returns) in the text object.

ATTR_GEN_TEXT_UPPERCASE_ALPHA allows alphabetic characters and spaces. ATTR_GEN_TEXT_UPPERCASE_ALPHA_NUMERIC, an allows only uppercase alphabetic characters and spaces. ATTR_GEN_TEXT_DASHED_ALPHA_NUMERIC allows only uppercase alphabetic characters and spaces but also allows dashed.

10.6

ATTR_GEN_TEXT_NORMAL_ASCII allows only normal ASCII characters (no extended ASCII characters).

ATTR_GEN_TEXT_LEGAL_DOS_VOLUME_NAMES allows only characters that may appear in a legal DOS volume name.

ATTR_GEN_TEXT_DOS_CHARACTER_SET allows only characters in the legal DOS character set.

ATTR_GEN_TEXT_NO_SPACES and ATTR_GEN_TEXT_ALLOW_SPACES govern whether spaces are allowed in the text object.

ATTR_GEN_TEXT_ALPHA, ATTR_GEN_TEXT_NUMERIC, ATTR_GEN_TEXT_SIGNED_NUMERIC, ATTR_GEN_TEXT_SIGNED_DECIMAL, ATTR_GEN_TEXT_FLOAT_DECIMAL, and ATTR_GEN_TEXT_ALPHA_NUMERIC govern the character types allowed.

ATTR_GEN_TEXT_LEGAL_FILENAMES allows any characters legal to file names.

ATTR_GEN_TEXT_NEVER_MAKE_SCROLLABLE forces the text object to expand larger as the text grows without forcing it into scrollable text.

ATTR_GEN_TEXT_DOES_NOT_ACCEPT_INK indicates that this text object will not accept ink under any circumstances.

ATTR_GEN_TEXT_CUSTOM_FILTER contains the ChunkHandle of an array of **VisTextCustomFilterData** structures. Each of these structures contains a range of character *values* to filter out. These values are determined from the **Chars** enumerated type. The text object figures out the number of filters in place by getting the size of the chunk.

◆Objects

Code Display 10-23 GenText Hints

```
@vardata ColorQuad       HINT_TEXT_WASH_COLOR
@vardata void            HINT_TEXT_WHITE_WASH_COLOR;
@vardata void            HINT_TEXT_AUTO_HYPHENATE;
@vardata void            HINT_TEXT_SELECT_TEXT;
@vardata void            HINT_TEXT_CURSOR_AT_START;
@vardata void            HINT_TEXT_CURSOR_AT_END;
@vardata void            HINT_TEXT_FRAME;
@vardata void            HINT_TEXT_NO_FRAME;
@vardata void            HINT_TEXT_ALLOW_UNDO;
@vardata void            HINT_TEXT_ALLOW_SMART_QUOTES;
@vardata void            HINT_TEXT_DO_NOT_MAKE_LARGER_ON_PEN_SYSTEMS;
```

HINT_TEXT_WASH_COLOR indicates the background color for the GenText to exhibit. This hint takes a **ColorQuad** argument.

HINT_TEXT_WHITE_WASH_COLOR indicates the background color of the GenText should be white.

10.6.2 GenText Basics

The GenText instance fields can be set to specific values in your Goc files. They may also be modified by your application at run-time. These instance fields may be modified by both **GenTextClass** messages or **VisTextClass** messages.

GenTextClass always builds out into a completely functioning subclass of **VisTextClass**. With the exception of geometry management behavior, you can assume that they will behave the same. You may send any **VisTextClass** message to a GenText object. Messages are only provided in **GenTextClass** to manipulate and alter behavior solely of **GenTextClass** origins.

10.6.2.1 VisText Instance Fields

The chunk handle stored in *GTXI_text* is copied to the VisText instance field *VTI_text* at run-time. To perform operations on this text, you may use any of the **VisTextClass** messages in “The Text” on page 614. Similarly, the text object maximum length stored in *GTXI_maxLength* is copied to *VTI_maxLength* at run-time. To perform operations on this data, use an appropriate message under “Using VisText” on page 653.

10.6.2.2 GenText Attributes

```
GTXI_attrs, MSG_GEN_TEXT_SET_ATTRS, MSG_GEN_TEXT_GET_ATTRS
```

GTXI_attrs stores the **GenTextAttrs** of the GenText object. These attribute flags are listed below.

GTA_SINGLE_LINE_TEXT
This flag indicates that this text object is only one line high. Vertical scrolling is disabled. Scrolling may be implemented horizontally if this is set.

GTA_USE_TAB_FOR_NAVIGATION
This flag indicates that the *Tab* key is used for navigation purposes in your application and should be interpreted to move to the next field rather than inserted into the text object.

GTA_INIT_SCROLLING
This flag indicates that the text object should appear with initial scrollbars.

GTA_NO_WORD_WRAPPING
This flag disables word wrapping.

GTA_ALLOW_TEXT_OFF_END
This flag is set if text may be allowed to overflow past the end of the text box. The text will still be stored in the object’s text chunk, but it will not force horizontal (or vertical) scrolling to show the text on-screen.

GTA_TAIL_ORIENTED
This flag is set if you prefer to display the tail end of text rather than the top end. In a scrolling text box, this ensures that the text being added at the end is always displayed.

GTA_DONT_SCROLL_TO_CHANGES
This flag disables the default behavior of scrolling to any changes being made within the text object. Normally, insertion or deletion of text will force a scrolling GenText to scroll to the point of action; this flag will turn this behavior off.

You may alter the contents of a GenText's *GTXI_attrs* instance field at run-time by sending the object a MSG_GEN_TEXT_SET_ATTRS. You may only send this message to a non-usable (~GS_USABLE) text object. To retrieve the current **GenTextAttrs** in use, send the text object MSG_GEN_TEXT_GET_ATTRS.

■ MSG_GEN_TEXT_SET_ATTRS

```
void        MSG_GEN_TEXT_SET_ATTRS(
            byte   attrsToSet,
            byte   attrsToClear);
```

This message sets a GenText object's **GenTextAttrs** (*GTXI_attrs*). The GenText object must not be GS_USABLE when sent this message.

Source: Unrestricted.

Destination: Any non-usable GenText object.

Parameters: *attrsToSet* **GenTextAttrs** to add.

attrsToClear **GenTextAttrs** to remove. An attribute set in both parameters will be cleared.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_TEXT_GET_ATTRS

```
byte        MSG_GEN_TEXT_GET_ATTRS();
```

This message retrieves the GenText object's **GenTextAttrs** (*GTXI_attrs*).

Source: Unrestricted.

Destination: Any GenText object.

Parameters: None.

Return: **GenTextAttrs** in use by the GenText object(*GTXI_attrs*).

Interception: Generally not intercepted.

10.6.2.3 GenTextStates

`GTXI_stateFlags, MSG_GEN_TEXT_SET_INDETERMINATE_STATE, MSG_GEN_TEXT_SET_MODIFIED_STATE, MSG_GEN_TEXT_IS_INDETERMINATE, MSG_GEN_TEXT_IS_MODIFIED`

GTXI_stateFlags stores the current state of the GenText object. There are two **GenTextStateFlags**:

GTSF_INDETERMINATE
: This flag specifies that the text within the GenText is indeterminate (may or may not reflect the current state). In most cases, you will not need to set this flag.

GTSF_MODIFIED
: This flag specifies that the text within the GenText has changed since it last received a MSG_GEN_APPLY. The handler for MSG_GEN_APPLY checks whether this flag is set before sending out the GenText's *GTXI_applyMsg*.

GenText objects should normally be marked by the application as not modified anytime their state is set with an external message. They will automatically be marked modified whenever the user interacts with them and marked not modified after receiving MSG_GEN_APPLY.

You may set a GenText's indeterminate or modified state with MSG_GEN_TEXT_SET_INDETERMINATE_STATE or MSG_GEN_TEXT_SET_MODIFIED_STATE, respectively. To check whether a GenText is indeterminate or modified, use MSG_GEN_TEXT_IS_INDETERMINATE or MSG_GEN_TEXT_IS_MODIFIED.

■ MSG_GEN_TEXT_SET_INDETERMINATE_STATE

```
void        MSG_GEN_TEXT_SET_INDETERMINATE_STATE(
            Boolean indeterminateState);
```

This message sets the GenText object's indeterminate state (the GTSF_INDETERMINATE flag in *GTXI_stateFlags*). This message does not affect the stored text.

Source: Unrestricted.

Destination: Any GenText object.

Parameters: *indeterminateState*
True to set the text object indeterminate (GTSF_INDETERMINATE); *false* otherwise.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_TEXT_SET_MODIFIED_STATE

```
void    MSG_GEN_TEXT_SET_MODIFIED_STATE(
        Boolean modifiedState);
```

This message allows you to set the modified state of a GenText object (the GTSF_MODIFIED flag in *GTXI_stateFlags*). This message does not affect the stored text.

Source: Unrestricted.

Destination: Any GenText object.

Parameters: *modifiedState* *True* to set the text object modified (GTSF_MODIFIED); *false* otherwise.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_TEXT_IS_INDETERMINATE

```
Boolean MSG_GEN_TEXT_IS_INDETERMINATE();
```

This message checks whether a GenText object is indeterminate in state.

Source: Unrestricted.

Destination: Any GenText object.

Parameters: None.

Return: The indeterminate state of the GenText object (*true* if the object is indeterminate, *false* otherwise).

Interception: Generally not intercepted.

■ MSG_GEN_TEXT_IS_MODIFIED

```
Boolean MSG_GEN_TEXT_IS_MODIFIED();
```

This message checks whether a GenText object has been modified.

Source: Unrestricted.

Destination: Any GenText object.

Parameters: None.

Return: The modified state of the GenText object (*true* if the object has been modified, *false* otherwise).

Interception: Generally not intercepted.

10.6.2.4 Sending an Action

`GTXI_applyMsg, GTXI_destination, MSG_GEN_TEXT_GET_DESTINATION, MSG_GEN_TEXT_SET_DESTINATION, MSG_GEN_TEXT_GET_APPLY_MSG, MSG_GEN_TEXT_SET_APPLY_MSG`

GTXI_applyMsg holds the message for the GenText to send out whenever it has been modified and needs to apply its changes. Whenever a GenText receives MSG_GEN_APPLY, it checks whether its GTSF_MODIFIED flag has been set; if it has, it will send out its apply message. If a GenText is operating in immediate mode, these actions will happen immediately, resulting in an immediate action.

Use the prototype GEN_TEXT_APPLY_MSG to define your apply message. This ensures that the apply message passes the correct parameters (the current **GenTextStateFlags**). *GTXI_destination* specifies the destination object (or process) to send the *GTXI_applyMsg* to. (This may also be a **TravelOption** such as TO_APP_FOCUS.) Use of these two fields is shown in Code Display 10-24.

Code Display 10-24 Sending an Apply Message

```
        /* Define the apply message using the provided prototype. */
@message (GEN_TEXT_APPLY_MSG) MSG_MY_TEXT_STUFF_TEXT_IN_BUFFER;

        /* In the object declaration, set the destination and the apply message. */
@object GenTextClass MyText = {
    GI_visMoniker = “My Text Object”;
    GTXI_text = “Initial Text Here”;
    GTXI_maxLength = 99;
    GTXI_applyMsg = MSG_MY_TEXT_STUFF_TEXT_IN_BUFFER;
    GTXI_destination = process;
}
```

```
/* Retrieve the current text and place it in the indicated buffer. */

@method MyTextProcessClass, MSG_MY_TEXT_STUFF_TEXT_IN_BUFFER {
    char tempBuffer[100];

/* The GET_ALL_PTR retrieves the current text and stuffs at the location it in the
 * passed pointer. Pass a text length of zero for null-terminated text strings. */
    @call MyText::MSG_VIS_TEXT_GET_ALL_PTR(tempBuffer);
}
```

To change a GenText's apply message or destination, send it MSG_GEN_TEXT_SET_APPLY_MSG or MSG_GEN_TEXT_SET_DESTINATION, respectively. Use MSG_GEN_TEXT_GET_APPLY_MSG or MSG_GEN_TEXT_GET_DESTINATION to return the current apply message or destination.

■ MSG_GEN_TEXT_SET_APPLY_MSG

```
void    MSG_GEN_TEXT_SET_APPLY_MSG(
        Message             message);
```

This message sets a new apply message (*GTXI_applyMsg*) for the text object.

Source: Unrestricted.

Destination: Any GenText object.

Parameters: *message* The new apply message.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_TEXT_GET_APPLY_MSG

```
Message MSG_GEN_TEXT_GET_APPLY_MSG();
```

This message retrieves the current apply message (*GTXI_applyMsg*) of a GenText object.

Source: Unrestricted.

Destination: Any GenText object.

Parameters: None.

Return: The apply message of the GenText object.

Interception: Generally not intercepted.

■ MSG_GEN_TEXT_SET_DESTINATION

```
void    MSG_GEN_TEXT_SET_DESTINATION(
        optr   dest);
```

This message sets the destination object or process (*GTXI_destination*) of a GenText object.

Source: Unrestricted.

Destination: Any GenText object.

Parameters: *dest* The optr of the new destination object or process.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_TEXT_GET_DESTINATION

```
optr    MSG_GEN_TEXT_GET_DESTINATION();
```

This message returns the current destination object (*GTXI_destination*) of a GenText object.

Source: Unrestricted.

Destination: Any GenText object.

Parameters: None.

Return: The optr of the GenText's destination object (*GTXI_destination*).

Interception: Generally not intercepted.

■ GEN_TEXT_APPLY__MSG Prototype

```
void    GEN_TEXT_APPLY_MSG(
        word   stateFlags);
```

This prototype should be used to define your GenText's apply message (with *GTXI_applyMsg*).

Parameters: *stateFlags* The text object's *GTXI_stateFlags* status.

10.6.2.5 Status Messages

ATTR_GEN_TEXT_STATUS_MSG, MSG_GEN_TEXT_SEND_STATUS_MSG

If your GenText is operating in delayed mode, there usually occur times when its current state may not reflect the most recent changes. In most cases, this is fine, but in some cases you may wish to notify other UI objects of a change in your GenText's state without sending out an apply message. This can be done with a *status message*.

A status message allows your GenText object to send out a message whenever the user interacts with the text object, regardless of whether that change will be immediately applied. This is most useful for cases in which two UI objects depend on each other. The status message allows one UI object to inform its counterpart that its state has changed, and that the counterpart should change its state to reflect the new information.

To give a GenText object a status message, include ATTR_GEN_TEXT_STATUS_MSG in the object's declaration. Use the prototype GEN_TEXT_STATUS_MSG to define your status message. This prototype ensures that the status message passes the correct parameters (the current **GenTextStateFlags**).

Any user changes that do not result in the sending of the object's apply message will result in the sending of the object's status message. For an object in immediate mode, this attribute will have no effect. You may also manually send an object's status message by sending the GenText object MSG_GEN_TEXT_SEND_STATUS_MSG.

■ MSG_GEN_TEXT_SEND_STATUS_MSG

```
void        MSG_GEN_TEXT_SEND_STATUS_MSG(
            Boolean             modifiedState);
```

This message causes a GenText object to send out its status message, stored in the text object's ATTR_GEN_TEXT_STATUS_MSG vardata field. This message will still function even if the text object is not enabled (or usable).

Source: Unrestricted.

Destination: Any GenText object.

Parameters: *modifiedState* Non-zero if GVSF_MODIFIED bit should be passed with the status message.

Return: Nothing.

Interception: Generally not intercepted.

■ **GEN_TEXT_STATUS__MSG** Prototype

```
void      GEN_TEXT_STATUS_MSG(
          word   stateFlags);
```

This prototype should be used to define your GenText's status message (with ATTR_GEN_TEXT_STATUS_MSG).

Parameters: *stateFlags* The text object's *GTXI_stateFlags* status.

10.7 The Controllers

This section describes the various text controller classes and how best to use them. Controllers create their UI gadgetry wherever in the generic tree they are placed; this section will give some pointers to designing the text controller UI. For full information on **GenControlClass** and how controllers work, see "Generic UI Controllers," Chapter 12.

10.7.1 Character Attribute Controllers

The following controllers affect character attributes within a text object. As such, they each affect the contents of a text object's character attributes instance field (usually the **VisTextCharAttr** structure).

TextStyleControlClass

This controller works with the style character attributes of the current text selection. The features and tools of this controller are shown in Code Display 10-25.

Code Display 10-25 TextStyleControlClass Features

```
typedef WordFlags TSCFeatures;
#define TSCF_PLAIN                      0x0800
#define TSCF_BOLD                       0x0400
#define TSCF_ITALIC                     0x0200
#define TSCF_UNDERLINE                  0x0100
#define TSCF_STRIKE_THRU                0x0080
```

```
#define TSCF_SUBSCRIPT                    0x0040
#define TSCF_SUPERSCRIPT                  0x0020
#define TSCF_BOXED                        0x0010
#define TSCF_BUTTON                       0x0008
#define TSCF_INDEX                        0x0004
#define TSCF_ALL_CAP                      0x0002
#define TSCF_SMALL_CAP                    0x0001

typedef WordFlags TSCToolboxFeatures;
#define TSCTF_PLAIN                       0x0800
#define TSCTF_BOLD                        0x0400
#define TSCTF_ITALIC                      0x0200
#define TSCTF_UNDERLINE                   0x0100
#define TSCTF_STRIKE_THRU                 0x0080
#define TSCTF_SUBSCRIPT                   0x0040
#define TSCTF_SUPERSCRIPT                 0x0020
#define TSCTF_BOXED                       0x0010
#define TSCTF_BUTTON                      0x0008
#define TSCTF_INDEX                       0x0004
#define TSCTF_ALL_CAP                     0x0002
#define TSCTF_SMALL_CAP                   0x0001

#define TSC_DEFAULT_FEATURES     (TSCF_PLAIN | TSCF_BOLD | TSCF_ITALIC |
                                  TSCF_UNDERLINE | TSCF_SUBSCRIPT |
                                  TSCF_SUPERSCRIPT | TSCF_STRIKE_THRU |
                                  TSCF_BOXED | TSCF_BUTTON | TSCF_INDEX |
                                  TSCF_ALL_CAP | TSCF_SMALL_CAP)
#define TSC_DEFAULT_TOOLBOX_FEATURES (TSCTF_PLAIN | TSCTF_BOLD | TSCTF_ITALIC |
                                 TSCTF_UNDERLINE)
```

FontControlClass

This controller allows the user to change font types of the current text selection. Its features and tools are shown in Code Display 10-26.

Code Display 10-26 FontControlClass Features

```
typedef WordFlags FCFeatures;
#define FCF_SHORT_LIST          0x0002
#define FCF_LONG_LIST           0x0001
```

```
typedef WordFlags FCToolboxFeatures;
#define FCTF_ROMAN              0x0004
#define FCTF_SANS               0x0002
#define FCTF_MONO               0x0001

#define FC_DEFAULT_FEATURES     (FCF_SHORT_LIST | FCF_LONG_LIST)
#define FC_DEFAULT_TOOLBOX_FEATURES (FCTF_ROMAN | FCTF_SANS | FCTF_MONO)
```

10.7 PointSizeControlClass

This controller controls the point size of the current text selection. Its features and tools are shown in Code Display 10-27.

Code Display 10-27 PointSizeControlClass Features

```
typedef WordFlags PSCFeatures;
#define PSCF_9                          0x0800
#define PSCF_10                         0x0400
#define PSCF_12                         0x0200
#define PSCF_14                         0x0100
#define PSCF_18                         0x0080
#define PSCF_24                         0x0040
#define PSCF_36                         0x0020
#define PSCF_54                         0x0010
#define PSCF_72                         0x0008
#define PSCF_SMALLER                    0x0004
#define PSCF_LARGER                     0x0002
#define PSCF_CUSTOM_SIZE                0x0001

typedef WordFlags PSCToolboxFeatures;
#define PSCTF_9                         0x0400
#define PSCTF_10                        0x0200
#define PSCTF_12                        0x0100
#define PSCTF_14                        0x0080
#define PSCTF_18                        0x0040
#define PSCTF_24                        0x0020
#define PSCTF_36                        0x0010
#define PSCTF_54                        0x0008
#define PSCTF_72                        0x0004
#define PSCTF_SMALLER                   0x0002
#define PSCTF_LARGER                    0x0001
#define PSC_DEFAULT_FEATURES            (PSCF_9 | PSCF_10 | PSCF_12 | PSCF_14 |
                                          PSCF_18 | PSCF_24 | PSCF_36 | PSCF_72 |
                                          PSCF_CUSTOM_SIZE | PSCF_SMALLER |
                                          PSCF_LARGER)
#define PSC_DEFAULT_TOOLBOX_FEATURES     (PSCTF_9 | PSCTF_10 | PSCTF_12 |
                                          PSCTF_14 | PSCTF_18 | PSCTF_24 |
                                          PSCTF_36 | PSCTF_72 | PSCTF_SMALLER |
                                          PSCTF_LARGER)
```

10.7 CharFGColorControlClass and CharBGColorControlClass

These controllers handle changes to the foreground (text) and background colors of the current text selection. They are both subclassed from **ColorSelectorClass** and therefore inherit the feature set of that class. The additional definitions for these classes are given in Code Display 10-28.

Code Display 10-28 Character Color Controller Features

```
/* Both CharFGColorControlClass and CharBGColorControlClass are subclasses of
 * ColorSelectorClass. For full details, see that class. */

@class CharFGColorControlClass, ColorSelectorClass;
#define CFGCC_DEFAULT_FEATURES   (CSF_INDEX | CSF_RGB | CSF_DRAW_MASK |
                                  CSF_PATTERN)
    @default GI_states = @default & ~GS_ENABLED;
    @default GCI_output = (TO_APP_TARGET);
@endc

@class CharBGColorControlClass, ColorSelectorClass;
#define CBGCC_DEFAULT_FEATURES   (CSF_INDEX | CSF_RGB | CSF_DRAW_MASK |
                                  CSF_PATTERN)
    @default GI_states = @default & ~GS_ENABLED;
    @default GCI_output = (TO_APP_TARGET);
@endc
```

FontAttrControlClass

This controller controls the font weight, font width, and track kerning of characters of the current text selection. Its features are shown in Code Display 10-29.

Code Display 10-29 FontAttrControlClass Features

```
typedef WordFlags FACFeatures;
#define FACF_FONT_WEIGHT        0x0004
#define FACF_FONT_WIDTH         0x0002
#define FACF_TRACK_KERNING      0x0001

typedef WordFlags FACToolboxFeatures;

#define FAC_DEFAULT_FEATURES     (FACF_FONT_WEIGHT | FACF_FONT_WIDTH |
                                  FACF_TRACK_KERNING)
#define FAC_DEFAULT_TOOLBOX_FEATURES 0
    @default GCI_output = (TO_APP_TARGET);
```

10.7.2 Paragraph Attribute Controllers

The following controllers affect paragraph attributes within a text object. As such, they each affect the contents of a text object's paragraph attributes instance field (usually the **VisTextParaAttr** structure).

JustificationControlClass

This controller allows the user to specify paragraph justification of the current text selection (left, right, center, or full). Its features and tools are shown in Code Display 10-30.

Code Display 10-30 JustificationControlClass Features

```
typedef WordFlags JCFeatures;
#define JCF_LEFT                0x0008
#define JCF_RIGHT               0x0004
#define JCF_CENTER              0x0002
#define JCF_FULL                0x0001

typedef WordFlags JCToolboxFeatures;
#define JCTF_LEFT               0x0008
#define JCTF_RIGHT              0x0004
#define JCTF_CENTER             0x0002
#define JCTF_FULL               0x0001
#define JC_DEFAULT_FEATURES     (JCF_LEFT | JCF_RIGHT | JCF_CENTER | JCF_FULL)
```

◆Objects

```
#define JC_DEFAULT_TOOLBOX_FEATURES (JCTF_LEFT | JCTF_RIGHT | JCTF_CENTER |
                                  JCTF_FULL)
    @default GCI_output = (TO_APP_TARGET);
    @default GI_attrs = (@default | GA_KBD_SEARCH_PATH);
```

ParaSpacingControlClass

10.7

This controller controls several paragraph spacing characteristics of the current text selection (such as paragraph spacing on top, spacing on bottom, leading, and line spacing). Its features and tools are shown in Code Display 10-31.

Code Display 10-31 ParaSpacingControlClass Features

```
typedef WordFlags PASCFeatures;
#define PASCF_SPACE_ON_TOP_BOTTOM        0x0002
#define PASCF_LEADING                    0x0001

typedef WordFlags PASCToolboxFeatures;
#define PASCTF_SINGLE                    0x8000
#define PASCTF_ONE_AND_A_HALF            0x4000
#define PASCTF_DOUBLE                    0x2000
#define PASCTF_TRIPLE                    0x1000

#define PASC_DEFAULT_FEATURES    (PASCF_SPACE_ON_TOP_BOTTOM | PASCF_LEADING)
#define PASC_DEFAULT_TOOLBOX_FEATURES (PASCTF_SINGLE | PASCTF_ONE_AND_A_HALF |
                                PASCTF_DOUBLE | PASCTF_TRIPLE)
@default GCI_output = (TO_APP_TARGET);
```

DefaultTabsControlClass and TabControlClass

These controllers set the placement and features of both default and user-defined tabs in the current text selection. The features and tools of both controllers are shown in Code Display 10-32.

Code Display 10-32 Tab Controllers

```
        /* DefaultTabsControlClass */
```

```
typedef WordFlags DTCFeatures;
#define DTCF_LIST               0x0002
#define DTCF_CUSTOM             0x0001

typedef WordFlags DTCToolboxFeatures;

#define DTC_DEFAULT_FEATURES    (DTCF_LIST | DTCF_CUSTOM)
#define DTC_DEFAULT_TOOLBOX_FEATURES 0
    @default GCI_output = (TO_APP_TARGET);

10.7    /* TabControlClass */

typedef WordFlags TCFeatures;
#define TCF_LIST                        0x0080
#define TCF_POSITION                    0x0040
#define TCF_GRAY_SCREEN                 0x0020
#define TCF_TYPE                        0x0010
#define TCF_LEADER                      0x0008
#define TCF_LINE                        0x0004
#define TCF_CLEAR                       0x0002
#define TCF_CLEAR_ALL                   0x0001

typedef WordFlags TCToolboxFeatures;

#define TC_DEFAULT_FEATURES     (TCF_LIST | TCF_POSITION | TCF_GRAY_SCREEN |
                                 TCF_TYPE | TCF_LEADER | TCF_LINE | TCF_CLEAR |
                                 TCF_CLEAR_ALL)
#define TC_DEFAULT_TOOLBOX_FEATURES 0
    @instance word TCI_numberOfTabs;
    @instance Tab TCI_tabList[VIS_TEXT_MAX_TABS];
    @instance word TCI_selectedTab;
    @default GCI_output = (TO_APP_TARGET);
```

ParaBGColorControlClass

This controller controls the background color, draw mask, and pattern of the background color of the current paragraph selections. Since **ParaBGColorControlClass** is a subclass of **ColorSelectorClass**, it inherits the functions of that class. Its features and tools are shown in Code Display 10-33.

Code Display 10-33 ParaBGColorControlClass Features

```
#define PBGCC_DEFAULT_FEATURES   (CSF_INDEX | CSF_RGB | CSF_DRAW_MASK |
                                  CSF_PATTERN)
    @default GI_states = @default & ~GS_ENABLED;
    @default GCI_output = (TO_APP_TARGET);
```

ParaAttrControlClass 10.7

This controller the user to set the “keep with,” hidden, word wrapping, an widow and orphan attributes of the current paragraph selection. Its features and tools are shown in Code Display 10-34.

Code Display 10-34 ParaAttrControlClass Features

```
typedef WordFlags PACFeatures;
#define PACF_WORD_WRAP                   0x0010
#define PACF_HIDDEN_TEXT                 0x0008
#define PACF_KEEP_PARA_WITH_NEXT         0x0004
#define PACF_KEEP_PARA_TOGETHER          0x0002
#define PACF_KEEP_LINES                  0x0001

typedef WordFlags PACToolboxFeatures;

#define PAC_DEFAULT_FEATURES     (PACF_WORD_WRAP | PACF_HIDDEN_TEXT |
                                  PACF_KEEP_PARA_WITH_NEXT |
                                  PACF_KEEP_PARA_TOGETHER | PACF_KEEP_LINES
#define PAC_DEFAULT_TOOLBOX_FEATURES 0
    @default GCI_output = (TO_APP_TARGET);
```

BorderControlClass and BorderColorControlClass

This controllers set border color and placement. **BorderColorControlClass** is a subclass of **ColorSelectorClass** and therefore inherits the features and tools of that class. The tools and features of both controllers are shown in Code Display 10-35.

Code Display 10-35 Border Controller Features

```
        /* BorderControlClass */

typedef WordFlags BCFeatures;
#define BCF_LIST                0x0002
#define BCF_CUSTOM              0x0001

typedef WordFlags BCToolboxFeatures;

10.7 #define BC_DEFAULT_FEATURES      (BCF_LIST | BCF_CUSTOM)
#define BC_DEFAULT_TOOLBOX_FEATURES 0
    @default GCI_output = (TO_APP_TARGET);

        /* BorderColorControlClass */

typedef WordFlags BCCToolboxFeatures;
#define BCC_DEFAULT_FEATURES    (CSF_INDEX | CSF_RGB | CSF_DRAW_MASK |
                                 CSF_PATTERN)
#define BCC_DEFAULT_TOOLBOX_FEATURES 0
    @default GI_states = @default & ~GS_ENABLED;
    @default GCI_output = (TO_APP_TARGET);
```

DropCapControlClass

This controller allows the user to set “drop cap” features of the current paragraph selection. Its features and tools are shown in Code Display 10-36.

Code Display 10-36 DropCapControlClass Features

```
typedef WordFlags DCCFeatures;
#define DCCF_DROP_CAP           0x0001

typedef WordFlags DCCToolboxFeatures;

#define DCC_DEFAULT_FEATURES    (DCCF_DROP_CAP)
#define DCC_DEFAULT_TOOLBOX_FEATURES 0
    @default GCI_output = (TO_APP_TARGET);
```

Objects

HyphenationControlClass

This controller sets the implementation of automatic word hyphenation of the current paragraph selection. Its features and tools are shown in Code Display 10-37.

Code Display 10-37 HyphenationControlClass

```
typedef WordFlags HCFeatures;
#define HCF_LIST                0x0001

typedef WordFlags HCToolboxFeatures;

#define HC_DEFAULT_FEATURES     (HCF_LIST)
#define HC_DEFAULT_TOOLBOX_FEATURES 0
    @default GCI_output = (TO_APP_TARGET);
```

MarginControlClass

This controller sets the left, paragraph, and right margins of the current paragraph selection. Its tools and features are shown in Code Display 10-38.

Code Display 10-38 MarginControlClass

```
typedef WordFlags MCFeatures;
#define MCF_LEFT_MARGIN         0x0004
#define MCF_PARA_MARGIN         0x0002
#define MCF_RIGHT_MARGIN        0x0001

typedef WordFlags MCToolboxFeatures;

#define MC_DEFAULT_FEATURES     (MCF_LEFT_MARGIN | MCF_PARA_MARGIN |
                                 MCF_RIGHT_MARGIN)
#define MC_DEFAULT_TOOLBOX_FEATURES 0
    @default GCI_output = (TO_APP_TARGET);
```

TextRulerControlClass

This controller manages how the **TextRulerClass** ruler operates, if any text ruler is used. The TextRuler provides a complete ruler for the text object. The features and tools of the TextRulerControl are shown in Code Display 10-39.

Code Display 10-39 TextRulerControlClass

```
typedef WordFlags TRCCFeatures;
#define TRCCF_ROUND                         0x0002
#define TRCCF_IGNORE_ORIGIN                 0x0001

typedef WordFlags TRCCToolboxFeatures;

#define TRCC_DEFAULT_FEATURES    (TRCCF_ROUND | TRCCF_IGNORE_ORIGIN)
#define TRCC_DEFAULT_TOOLBOX_FEATURES 0

typedef WordFlags TextRulerControlAttributes;
#define TRCA_ROUND                          0x8000
#define TRCA_IGNORE_ORIGIN                  0x4000
    @instance TextRulerControlAttributes TRCI_attrs = TRCA_ROUND;
```

TextStyleSheetControlClass

This controller allows the user to set up and use style sheets with the target text object. It is subclassed off **StyleSheetControlClass** and inherits all the features and tools of that class. It has no additional tools or features of its own.

10.7.3 Search and Replace and Spell-Checking

`TextSearchInString(), TextSearchInHugeArray(), SearchOptions, WildCard`

Text searches are not specifically part of **VisTextClass**, but are commonly used on text objects, and they are discussed here for that reason.

TextSearchInString() searches in a single text chunk for a passed text string. If a match is found, a pointer to that match (and the length of the match) are returned in buffers. **TextSearchInHugeArray()** performs the same searching operations, but can return the segment and offset of a found match within a huge array.

The following **SearchOptions** may affect the search procedure:

SO_NO_WILDCARDS
If set, the search mechanism ignores any passed **WildCard** values and treats them as the control character values (CTRL_P, CTRL_Q and CTRL_R) which they overlap.

SO_IGNORE_SOFT_HYPHENS
If set, soft hyphens in the "searched-in" text are treated as if they do not exist, therefore allowing the text "hyphenation" to match "hyphen-ation" in the destination text. If the match string contains soft hyphens, do not set this flag as strings will never match.

SO_BACKWARD_SEARCH
If set, the search will propagate backwards.

SO_IGNORE_CASE
If set, case will be ignored in the search criteria.

SO_PARTIAL_WORD
If set, partial words will be matched.

SO_PRESERVE_CASE_OF_DOCUMENT_STRING
If set, the case of the match string will be altered to preserve the case of the before

Both routines may take wild cards (type **WildCard**) in the place of any character. These wildcards tell the search mechanism to accept any character in the allotted space. Passing WC_MATCH_MULTIPLE_CHAR instructs the search mechanism to accept any number of random characters in that place in the search string. The values of **WildCard** are:

WC_MATCH_SINGLE_CHAR
WC_MATCH_MULTIPLE_CHAR
WC_MATCH_WHITESPACE_CHAR

You can set higher-level features of your text objects with the **SearchReplaceControlClass** and **SpellControlClass** controllers. These controllers allow spell-checking and search and replace. These two controllers are designed to interact with each other and use common messages.

SearchReplaceControlClass

This controller allows the use of the text object's built-in search and replace features. The controller must be placed on the GAGCNLT_SELF_LOAD_OPTIONS Application GCN list. This controller sends out the following messages to its *GCI_output*, which may be intercepted to provide custom behavior:

MSG_SEARCH
MSG_REPLACE_CURRENT
MSG_REPLACE_ALL_OCCURENCES
MSG_REPLACE_ALL_OCCURENCES_IN_SELECTION
MSG_META_GET_OBJECT_FOR_SEARCH_SPELL
MSG_META_DISPLAY_OBJECT_FOR_SEARCH_SPELL

Because many of these messages are shared by the SpellControlClass, they are described in "Shared Functionality" on page 713.

The Search & Replace messages make use of a **SearchReplaceStruct**. This structure is followed by text string(s) that represent the strings to search (and replace if that is the case). These strings may contain **WildCard** values (unless SO_NO_WILDCARDS is set in *params*).The **SearchReplaceStruct** is shown below:

```
typedef struct {
      word        searchSize;
      word        replaceSize;
      byte        params;
      optr        replyObject;
      Message     replyMsg;
} SearchReplaceStruct;
```

searchSize stores the number of characters in the search string, including the null terminator.

replaceSize stores the number of characters in the replace string (if present), including the null terminator.

params stores the **SearchOptions** in use for this operation.

replyObject stores the optr of the object to send the *replyMsg* to if the string is not found.

replyMsg stores the message sent to the *replyObject* above.

The block containing this **SearchReplaceStruct** will contain either one null-terminated string (if the operation is just a simple search operation) or a null-terminated search string followed by a null-terminated replace string (if the operation is a search & replace).

■ MSG_SEARCH

```
@importMessage MetaSearchSpellMessages, void MSG_SEARCH(
          MemHandle           searchInfo);
```

This message is sent by the SearchReplace controller when an object is starting a search operation. You may intercept this message to find out the nature and specifics of the search.

Source: Search controller.

Destination: *GCI_output* of the search controller (usually a text object).

Parameters: *searchInfo* Handle of block containing the **SearchReplaceStruct**. The search string immediately follows this structure in the passed block. This block should be freed by the message handler.

Interception: Intercept to find out the nature of the search and replace operation.

The features and tools of this controller are shown in Code Display 10-40.

Code Display 10-40 SearchReplaceControlClass Features

```
typedef WordFlags SRCFeatures;
#define SRCF_CLOSE                        0x200
#define SRCF_FIND_NEXT                    0x100
#define SRCF_FIND_PREV                    0x80
#define SRCF_REPLACE_CURRENT              0x40
#define SRCF_REPLACE_ALL_IN_SELECTION     0x20
#define SRCF_REPLACE_ALL                  0x10
#define SRCF_PARTIAL_WORDS                0x08
#define SRCF_IGNORE_CASE                  0x04
#define SRCF_WILDCARDS                    0x02
#define SRCF_SPECIAL_CHARS                0x01
```

```
typedef WordFlags SRCToolboxFeatures;
#define SRCTF_SEARCH_REPLACE            0x01

#define SRC_DEFAULT_FEATURES     (SRCF_FIND_NEXT | SRCF_FIND_PREV |
                                  SRCF_REPLACE_CURRENT | SRCF_REPLACE_ALL |
                                  SRCF_PARTIAL_WORDS | SRCF_IGNORE_CASE |
                                  SRCF_WILDCARDS | SRCF_SPECIAL_CHARS |
                                  SRCF_REPLACE_ALL_IN_SELECTION | SRCF_CLOSE)
#define SRC_DEFAULT_TOOLBOX_FEATURES (SRCTF_SEARCH_REPLACE)
```

SpellControlClass

This controller allows you to include the advanced spelling-checker features of the text object. The controller must be placed on the GAGCNLT_SELF_LOAD_OPTIONS Application GCN list. This controller sends out the following messages, which may be intercepted to provide custom behavior:

MSG_SPELL_CHECK
MSG_REPLACE_CURRENT
MSG_REPLACE_ALL_OCCURENCES
MSG_META_GET_CONTEXT
MSG_META_CONTEXT
MSG_META_GET_OBJECT_FOR_SEARCH_SPELL
MSG_META_DISPLAY_OBJECT_FOR_SEARCH_SPELL

■ MSG_SPELL_CHECK

```
@importMessage MetaSearchSpellMessages, void MSG_SPELL_CHECK(@stack
          optr                 replyObj,
          dword                numCharsToCheck,
          SpellCheckOptions    options,
          MemHandle            ICbuff);
```

This message is sent by the Spell controller to continue spell checking from the current position in the document.

Source: Spell controller.

Destination: *GCI_output* of the spell controller.

Parameters: *replyObj* Object to send MSG_SPELL_CHECK_COMPLETED.

numCharsToCheck
Number of characters to check (if *options* has SCO_CHECK_NUM_CHARS set).

options **SpellCheckOptions** in use for this operation.

ICbuff Handle of buffer to pass spell check library.

Interception: Intercept to find out the nature of the spell checking operation.

■ MSG_META_GET_CONTEXT

```
@importMessage MetaSearchSpellMessages, void MSG_META_GET_CONTEXT(@stack
          dword                   position,
          ContextLocation         location,
          word                    numCharsToGet,
          optr                    replyObj);
```

This message is sent by the Spell controller to find out the current context of the text object. Other objects may also send this message to the text object. The text object will return MSG_META_CONTEXT to the requestor.

Source: The Spell controller. (Other objects may want to send this to retrieve the text context displayed in the spell box.)

Destination: *GCI_output* of the Spell controller (usually a text object).

Parameters: *position* Position of the context. The context returned depends on this value and the **ContextLocation** passed in *location*.

location **ContextLocation**. This may be one of the following values:

CL_STARTING_AT_POSITION
CL_ENDING_AT_POSITION
CL_CENTERED_AROUND_POSITION
CL_CENTERED_AROUND_SELECTION
CL_CENTERED_AROUND_SELECTION_START
CL_SELECTED_WORD

numCharsToGet Maximum number of characters to return.

replyObj Optr of the object to reply to via MSG_META_CONTEXT.

Interception: You may intercept to alter the nature of the context request.

■ MSG_META_CONTEXT

```
@importMessage MetaSearchSpellMessages, void MSG_META_CONTEXT(
                  MemHandle          data);
```

This message returns the current context to the SpellControl object.

Source: A text object receiving MSG_META_GET_CONTEXT.

Destination: The object that sent the request.

Parameters: *data* **ContextData** structure storing information on the context, and the null-terminated context string.

Structures:

```
typedef struct {
      optr                CD_object;
      dword               CD_numChars;
      dword               CD_startPos;
      VisTextRange        CD_selection;
} ContextData;
```

CD_object stores the optr of the object that the context is coming from.

CD_numChars stores the number of characters in the text object.

CD_range stores the range of characters that this context represents within the text object.

CD_selection stores the range of characters that represent the current text selection.

The null-terminated context data follows this structure.

Interception: Generally not intercepted.

Its feature and tool definitions are shown in Code Display 10-41.

Code Display 10-41 SpellControlClass

```
typedef WordFlags SpellFeatures;
#define SF_CLOSE                0x2000
#define SF_CONTEXT              0x1000
#define SF_SIMPLE_MODAL_BOX     0x0800
#define SF_SUGGESTIONS          0x0400
#define SF_CHECK_ALL            0x0200
#define SF_CHECK_TO_END         0x0100
```

```
#define SF_CHECK_SELECTION      0x0080
#define SF_SKIP                 0x0040
#define SF_SKIP_ALL             0x0020
#define SF_REPLACE_CURRENT      0x0010
#define SF_REPLACE_ALL          0x0008
#define SF_ADD_TO_USER_DICTIONARY0x0004
#define SF_EDIT_USER_DICTIONARY 0x0002
#define SF_STATUS               0x0001

typedef WordFlags SpellToolboxFeatures;
#define STF_SPELL               0x01

#define SC_DEFAULT_FEATURES             (SF_STATUS | SF_EDIT_USER_DICTIONARY |
                SF_ADD_TO_USER_DICTIONARY | SF_REPLACE_ALL | SF_REPLACE_CURRENT |
                SF_SKIP_ALL | SF_SKIP | SF_CHECK_SELECTION | SF_CHECK_TO_END |
                SF_SUGGESTIONS | SF_CLOSE | SF_CONTEXT)
#define SC_DEFAULT_TOOLBOX_FEATURES     STF_SPELL
```

Shared Functionality

The SearchReplace and Spell controllers share a common range of messages (**MetaSearchSpellMessages**) imported from **MetaClass**. (The messages MSG_SEARCH and MSG_SPELL_CHECK are specific to their respective controllers and were described earlier.) These messages common to both controllers are listed and described here.

■ MSG_REPLACE_CURRENT

```
@importMessage MetaSearchSpellMessages, void MSG_REPLACE_CURRENT(
          MemHandle           replaceInfo);
```

This message is sent by the SearchReplace controller when an object is starting a search operation that should replace the found match with the passed replacement string. This message involves a single replacement operation. To perform a replace-all operation, use MSG_REPLACE_ALL_OCCURENCES.

Source: Search controller.

Destination: *GCI_output* of the search controller (usually a text object).

Parameters: *replaceInfo* Handle of block containing the **SearchReplaceStruct**. The null-terminated search string immediately follows this structure.

The null-terminated replace string follows that string. This block should be freed by the message handler.

Interception: Intercept to find out the nature of the replacement operation.

■ MSG_REPLACE_ALL_OCCURRENCES

10.7

```
@importMessage MetaSearchSpellMessages, void
                MSG_REPLACE_ALL_OCCURRENCES(
          MemHandle           replaceInfo,
          Boolean             replaceFromBeginning);
```

This message is sent by the SearchReplace controller when an object is starting a search operation that should replace all found matches (in the search string) with the passed replacement string.

Source: Search controller.

Destination: *GCI_output* of the search controller (usually a text object).

Parameters: *replaceInfo* Handle of block containing the **SearchReplaceStruct**. The null-terminated search string immediately follows this structure. The null-terminated replace string follows that string. This block should be freed by the message handler.

replaceFromBeginning
Non-zero if the replace-all operation should proceed from the beginning of the document; normally, the replace-all operation begins after the current position.

Interception: Intercept to find out the nature of the replacement operation.

■ MSG_REPLACE_ALL_OCCURRENCES_IN_SELECTION

```
@importMessage MetaSearchSpellMessages, void
                MSG_REPLACE_ALL_OCCURRENCES_IN_SELECTION(
          MemHandle           replaceInfo,
          Boolean             replaceFromBeginning);
```

This message is sent by the SearchReplace controller when an object is starting a search operation that should replace all found matches within a selection (in the search string) with the passed replacement string.

Source: Search controller.

Destination: *GCI_output* of the search controller (usually a text object).

Parameters: *replaceInfo* Handle of block containing the **SearchReplaceStruct**. The null-terminated search string immediately follows this structure. The null-terminated replace string follows that string. This block should be freed by the message handler.

replaceFromBeginning Non-zero if the replace-all operation should proceed from the beginning of the selection; normally, the replace-all operation begins after the current position.

Interception: Intercept to find out the nature of the replacement operation.

■ MSG_META_GET_OBJECT_FOR_SEARCH_SPELL

```
@importMessage MetaSearchSpellMessages, void
                MSG_META_GET_OBJECT_FOR_SEARCH_SPELL(
          GetSearchSpellObjectOption    option,
          optr                          curObject);
```

This message is sent by a text object to itself when a search reaches the end of the current object and needs to determine where to continue the search.

Source: Any text object.

Destination: The text object sends this to itself.

Parameters: *option* **GetSearchSpellObjectOption** specifying the next object to continue the search at. This value may be GSSOT_FIRST_OBJECT, GSSOT_LAST_OBJECT, GSSOT_NEXT_OBJECT, or GSSOT_PREV_OBJECT.

curObject Optr of the object that is currently being searched through. If you intercept this message, you can use this information to set the *option* above.

Interception: May intercept if you want to direct where a search or spell check operation should continue to.

■ MSG_META_DISPLAY_OBJECT_FOR_SEARCH_SPELL

```
@importMessage MetaSearchSpellMessages, void
              MSG_META_DISPLAY_OBJECT_FOR_SEARCH_SPELL();
```

This message is sent by the text object to itself when a matching word (or misspelled word if the controller is a Spell controller) has been found and should be displayed.

Source: The text object.

Destination: The text object sends this to itself.

Interception: You may intercept this message at the text object to receive notification of a match.

The List Objects

11

The GenItemGroup and GenBooleanGroup objects manage lists of objects (GenItems or GenBooleans, respectively) that contain user-selectable options. **GenDynamicListClass** is a subclass of **GenItemGroupClass** which allows efficient dynamic and scrolling lists. All of the list objects manage children which each represent a particular option. The user can generally select and deselect options in such a group, and in doing so cause an action to take place.

The list groups and list items work together to manage states of objects within your application. These list objects perform all of the complex state management of their children; the selection items themselves are very simple objects.

You should be familiar with generic objects before reading this chapter. Please see “The GEOS User Interface,” Chapter 10 of the Concepts Book for an overview of each generic object in the user interface. You should be particularly familiar with the instance data of and messages handled by **GenClass**.

11.1 List Object Features

The list objects provide a highly flexible way to display and manage user-selectable options. Depending on the Specific UI and the type of list, these objects may be built out in a variety of fashions. In some UIs, BooleanGroups may appear as checkboxes, GenItemGroups may appear as radio buttons, etc. For examples in OSF/Motif, see Figure 11-1.

You may manipulate the grouping objects in much the same way as a GIV_SUB_GROUP GenInteraction. For example, hints may alter the placement and dimensions of the grouping object’s children or the list may appear with special dimensions. The items themselves are simple generic objects that can be easily added to your grouping object. The grouping object will manage the geometry and selection state of these children; it will also perform any required actions based on the state of these children.

There are three list management objects.

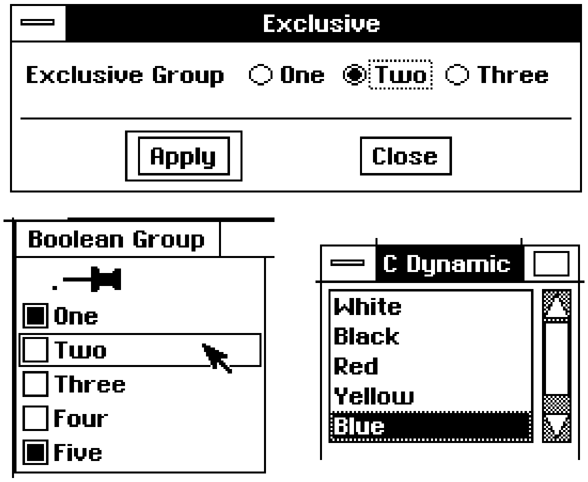


Figure 11-1 *Examples of several list objects*

Examples of a GenItemGroup (exclusive), GenBooleanGroup and GenDynamicList object.

- **GenItemGroup**
  An all-purpose grouping object, this list may be exclusive (only one item may be selected at a time) or non-exclusive and may also have scroll bars. A GenItemGroup may only contain **GenItemClass** objects as its children.

- **GenBooleanGroup**
  This list is designed for entries that are either true of false (“on” or “off”) independent of other entries. The BooleanGroup keeps track of its selections with a bitfield representing the boolean state of each of its children. A GenBooleanGroup object may only contain **GenBooleanClass** objects.

- **GenDynamicList**
  This list is designed for very long or changing length lists. In many respects it is similar to a GenItemGroup but also allows efficient dynamic lists. A GenDynamicList does not specify any explicit “children”; instead it contains a list of monikers to display to the user within a scrolling list.

These are the following main list item objects:

◆Objects

- **GenItem**
  Objects used as entries in a GenItemGroup. In OSF/Motif, these objects appear in most cases as “diamond” settings.
- **GenBoolean**
  Objects used as entries in a GenBooleanGroup representing true or false states. In OSF/Motif, these items appear most often as selectable “boxes.”

The GenDynamicList does not specify any generic children, but instead will contain a list of visual monikers to cycle through a scrolling list.

11.2 Common Behavior

Each list object acts differently and you will need to carefully review these differences before using them. There are a number of similarities in the functionality of these list objects, however. This next section details these similarities.

Each list group acts as a grouping object for its generic children. The list groups have much of the same geometry capabilities as a GIV_SUB_GROUP Interaction; they can arrange their children in a variety of fashions and can also be manipulated within their parent.

The items themselves may in general be either “on” (selected) or “off” (unselected). Depending on the behavior of the list, selecting an item may cause an action to occur and may affect the other selections. In exclusive lists, for example, selecting any item deselects any other item.

11.2.1 Applying the Action

Often, each list object will contain an “apply” message. This message will be sent out when the list’s state has changed and the new state should be applied in the application. For example, a list of colors (“Red”, “Green”, and “Blue”) may have a message MSG_GEN_VIEW_SET_COLOR that sets the background color of a GenView. Changing the state of the list (by selecting an item) will cause this notification message to be sent to the view. More typically, though, the apply message will be sent to the application or a controller object.

Alternatively, you could forego including an apply message in the list object's instance data. In this case, whenever the object receives a MSG_GEN_APPLY it would query the state of its children and act upon that state.

When the user makes a selection, the apply action may or may not be sent out immediately, depending on the operating mode of the list. GenItemGroups, GenBooleanGroups, and GenDynamicLists may operate in one of two modes: immediate or delayed. (For a more complete discussion of delayed and immediate modes, see "GenInteraction," Chapter 7.) Typically, a list object will operate in delayed mode if it is placed within a GIT_PROPERTIES Interaction. Other generic objects such as GenInteractions, GenTexts, and GenValues also utilize delayed and immediate mode.

In immediate mode, any changes made in the list's state are reflected immediately in the application. If the user changes an immediate mode list object concerning a text style change, the text style of the current selection will change immediately. Changing the state of the list will cause the apply action associated with that object to be sent out immediately.

In delayed mode, any changes made in the list's state are not reflected until some later time, when the changes are "applied." This apply may be effected by the user (through an "apply" trigger) or by the application when it deems it appropriate. In delayed mode, the notification message will not be sent on changes to the state of the list but only when the list receives notice to apply the changes.

Occasionally, a list operating in delayed mode may wish to send out a different message whenever the user changes its state, even if that change should not be immediately applied. In this case, you can supply a status message for the object to send to the destination whenever the user makes changes within the list group.

Each list must have a destination to send its apply action and, if needed, status messages to. This destination can be any object or process capable of handling the message.

11.2.2 State Information

In most cases, a selection will be either on or off. In some cases, however, a selection or a list's state will be indeterminate—considered neither completely on or off. For example, a styles list may show the characteristics of the current highlighted text (bold, italic, underlined, etc.). Selecting a whole paragraph of text in which only one word is bold will leave the "bold" entry in the styles list in an indeterminate state—some of the selected text is bold, some is not.

The lists keep track of these indeterminate states themselves. In some cases, an entire list may be indeterminate; in other cases, individual items may be indeterminate. It is up to the specific UI to represent an indeterminate state. (In OSF/Motif settings that are indeterminate are shown unselected.)

If a list object is operating in delayed mode, changes within the list will not be effected until either the user or the application "applies" them. The list objects keep track of changes in state by storing a modified state of either the list or individual items. Whenever an apply action is made on a list group, the group will check whether its list (or its items) have been modified. If nothing has been marked modified, the list will not apply those changes (unless you wish to override this default behavior).

This modified state is independent of the list's mode. In immediate mode, a change in state to a list object will automatically mark that list object modified and simultaneously call MSG_GEN_APPLY on the list object. The list object will recognize that it is modified and will send out the group's apply action. In delayed mode, any changes made to the object will also mark the group modified; at some later point, MSG_GEN_APPLY will be sent to the list; the object will recognize that the group has been modified and will then send out its apply action.

11.3 GenItemGroups

The **GenItemGroupClass** and **GenItemClass** work together to allow an application to set up a list of options for a user to choose. A GenItemGroup may contain one or more GenItems; each GenItem may represent a particular option in the list. The user may select one or more of the options at any time, depending upon the behavior specified for the list, and the user can generally select or deselect an option by activating the corresponding GenItem.

GenItemClass objects are simple list items. GenItems are used only as entries within a GenItemGroup. In most cases, a GenItem will only need a moniker and a special identifier associated with the selection. The GenItemGroup will use this identifier to refer to the object when it needs to communicate to the items or when it sends out messages.

Your application interacts with GenItems though the GenItemGroup; this group controls the items and keeps track of which are selected. The GenItemGroup object has all of the same geometry features as a grouping type GenInteraction. (See “GenInteraction,” Chapter 7).

11.3.1 GenItemGroup Instance Data

GenItemGroup and GenItem objects contain a variety of information. The bulk of the functionality of the list is contained in the GenItemGroup; the items themselves only contain identifying information that the list uses.

The GenItemGroup contains instance fields relating to the state of its children GenItems. These instance fields are listed in Code Display 11-1. Any objects of **GenItemGroupClass** or one of its subclasses will contain these instance fields, along with the instance fields of **GenClass**.

Code Display 11-1 GenItemGroup Instance Data

```
    @instance GenItemGroupBehaviorType   GIGI_behaviorType = GIGBT_EXCLUSIVE;

/* The GenItemGroupBehaviorType enumerations are used in the GIGI_behaviorType
 * instance field. */

typedef ByteEnum GenItemGroupBehaviorType;
#define GIGBT_EXCLUSIVE 0
#define GIGBT_EXCLUSIVE_NONE 1
#define GIGBT_EXTENDED_SELECTION 2
#define GIGBT_NON_EXCLUSIVE 3

    @instance GenItemGroupStateFlags     GIGI_stateFlags = 0;
```

```
/* The GenItemGroupStateFlags are used in the GIGI_stateFlags instance field. */

typedef ByteFlags GenItemGroupStateFlags;
#define GIGSF_INDETERMINATE      0x80
#define GIGSF_MODIFIED           0x40

    @instance word                      GIGI_selection = GIGS_NONE;

/* The GIGS_NONE constant is used with the GIGI_selection instance field and may
 * also be returned by MSG_GEN_ITEM_GROUP_GET_SELECTION. */

#define GIGS_NONE (0xffff)

    @instance word                      GIGI_numSelections = 0;
    @instance optr                      GIGI_destination;
    @instance Message                   GIGI_applyMsg = 0;
```

GIGI_behaviorType describes the selection behavior of the list. The default type is GIGBT_EXCLUSIVE, which creates an exclusive list. A full description of each type is provided in “GenItemGroup Behavior” on page 728.

GIGI_stateFlags contains flags that affect the state of the GenItemGroup. These flags indicate whether a GenItemGroup is in an indeterminate or a modified state.

GIGI_selection contains the name of the current GenItem child that is selected. GenItemGroups use the identifying keywords stored in each GenItem’s *GII_identifier* instance field to reference those selections. If more than one item may be selected at a time (if *GIGI_numSelections* is greater than one), this instance field will point to a list of GenItem identifiers rather than store a single identifier.

GIGI_numSelections contains the number of GenItems currently selected. For exclusive or exclusive-none lists, this number will be either one or zero. For multiple selection lists, this number may be greater than zero. Unless you have an item group that initially appears with more than one selection, you will not need to set this up in your object declaration.

GIGI_destination contains the optr of the object or process to handle messages sent out by this GenItemGroup. This destination object will receive the notification message in the *GIGI_applyMsg* instance field whenever state changes are applied within the GenItemGroup.

GIGI_applyMsg stores the apply action for the GenItemGroup to send out whenever state changes should be applied, either through an immediate state change or through a delayed mode activation.

The GenItemGroup also has several vardata fields.

Code Display 11-2 GenItemGroup Vardata Instance Fields

```
@vardata Message        ATTR_GEN_ITEM_GROUP_STATUS_MSG;
@vardata void           ATTR_GEN_ITEM_GROUP_SET_MODIFIED_ON_REDUNDANT_SELECTION;
@vardata Message        ATTR_GEN_ITEM_GROUP_CUSTOM_DOUBLE_PRESS;
@vardata optr           ATTR_GEN_ITEM_GROUP_LINK;
    @reloc ATTR_GEN_ITEM_GROUP_LINK, 0, optr;
@vardata void           ATTR_GEN_ITEM_GROUP_INIT_FILE_BOOLEAN;

/* Hints */

@vardata void   HINT_ITEM_GROUP_RADIO_BUTTON_STYLE;
@vardata void   HINT_ITEM_GROUP_TOOLBOX_STYLE;
@vardata void   HINT_ITEM_GROUP_SCROLLABLE;
@vardata void   HINT_ITEM_GROUP_MINIMIZE_SIZE;
@vardata void   HINT_ITEM_GROUP_DISPLAY_CURRENT_SELECTION;
@vardata void   HINT_ITEM_GROUP_MINIMIZE_SIZE_IF_VERTICAL_SCREEN;
```

ATTR_GEN_ITEM_GROUP_STATUS_MSG assigns a status message to your GenItemGroup. A status message allows objects to receive notice of a change in the GenItemGroup's current state without forcing an apply action. This attribute is useful for cases where the state of your GenItemGroup may reflect information in other user interface objects, and should therefore be updated whenever its state changes, regardless of the need to apply those changes.

In exclusive or extended-selection lists, selecting an already-select item will not change the state of the GenItemGroup and will not mark it modified. ATTR_GEN_ITEM_GROUP_SET_MODIFIED_ON_REDUNDANT_SELECTION alters this default behavior for a GenItemGroup. Including this attribute in the object's instance data will cause the group to be marked modified whenever the user makes a redundant selection, thereby forcing an apply action to later take place. Use this attribute for operations you may wish to repeat even if the state of the GenItemGroup has not changed.

ATTR_GEN_ITEM_GROUP_LINK allows two or more GenItemGroups to be linked together and act as one GenItemGroup. Each GenItemGroup may contain one or more items with identifiers unique to all other items within the linked GenItemGroups. The links should be circular, with the last group pointing back to the first. All of these GenItemGroups should have identical states and behavior types so they will work correctly. Use this attribute to circumvent undesirable geometry constraints. For more information, see “GenItemGroup Links” on page 752.

ATTR_GEN_ITEM_GROUP_CUSTOM_DOUBLE_PRESS allows a special activation message to be sent out on double press events; it is used only in exclusive GenBooleanGroup lists.

ATTR_GEN_ITEM_GROUP_INIT_FILE_BOOLEAN forces the GenItemGroup to use “true” and “false” in the .INI file rather than numerical values.

HINT_ITEM_GROUP_RADIO_BUTTON_STYLE indicates that the Item Group should appear as “radio buttons” if the Specific UI allows such behavior. OSF/Motif does this by default. HINT_ITEM_GROUP_TOOLBOX_STYLE indicates that the item’s should be drawn in toolbox style, with a simple box around the visual moniker.

HINT_ITEM_GROUP_MINIMIZE_SIZE and HINT_ITEM_GROUP_MINIMIZE_SIZE_IF_VERTICAL_SCREEN force the item group to use only the smallest space available. In OSF/Motif, for example, this would cause the item group to be a popup menu. The latter of the two hints takes effect only on screens that are taller than they are wide.

11.3.2 GenItem Instance Data

GenItemClass is a subclass of **GenClass**. It has one new instance field:

```
@instance word GII_identifier;
```

GII_identifier sets the indentifying keyword (usually an enumerated type) that uniquely identifies this GenItem. This identifier is used by the GenItemGroup parent to indicate the current selection and otherwise reference the item. Each GenItem identifier within the same GenItemGroup must evaluate to a unique value.

11.3.3 GenItemGroup Basics

The GenItemGroup manages a group of GenItems. In general, you must decide which behavior type you wish your GenItemGroup to exhibit, the message for the list to send out, and the object (or process) to handle messages sent out by this group. You may also set a number of initial conditions (items selected, whether the list is indeterminate or modified, etc.) for the list.

11.3.3.1 GenItemGroup Behavior

`GIGI_behaviorType, MSG_GEN_ITEM_GROUP_SET_BEHAVIOR_TYPE, MSG_GEN_ITEM_GROUP_GET_BEHAVIOR_TYPE`

There are several types of user selection behavior for a GenItemGroup. Items can be set up so that only one is selected at a time, one or none is selected at a time, multiple items can be selected, or multiple items can be selected only when extending the selection (e.g. dragging the mouse).

The *GIGI_behaviorType* instance field defines the selection behavior of the list. The instance data must be of the enumerated type **GenItemGroupBehaviorType**. The default type is GIGBT_EXCLUSIVE, which creates an exclusive list; one and only one selection in an exclusive list may be selected at any time. The available types are listed below.

GIGBT_EXCLUSIVE

This is a list where only one option may be selected at a time. Once a selection has been made, making another selection will deselect the previous item. The user may not deselect a currently selected item. Many specific UIs represent these items with radio buttons; OSF/Motif represents these items with “diamond” settings. This type of list is the default for a GenItemGroup.

GIGBT_EXCLUSIVE_NONE

This list is also exclusive—at most one option may be selected at a time—but the user can also deselect an item, leaving no items selected. GenItemGroups show a “none-selected” state with GIGS_NONE in their *GIGI_selection* instance fields.

GIGBT_EXTENDED_SELECTION

This list is normally exclusive, unless the user "extends" the selection. The user can extend the selection either by dragging the mouse through another selection or by clicking on a new selection in conjunction with a special keystroke. In OSF/Motif, clicking on an item while holding the SHIFT key will extend the selection (leaving any other selections unaffected) to the one clicked on. Extending a selection to a currently selected item will deselect that item.

GIGBT_NON_EXCLUSIVE

This list does not exhibit any exclusive behavior. Any or all settings may be on at any time, and selecting any items will not affect the state of other settings. In many case, you may want to use a GenBooleanGroup in place of a non-exclusive GenItemGroup. For example, a non-exclusive GenItemGroup cannot store individual indeterminate or modified states for its children; GenBooleanGroup stores this information.

Exclusive Lists

Exclusive lists are designed for cases in which one and only item should be selected at any given time. For example, a graphics application may want to prompt the user to select a color for drawing lines. The application can display this list of colors within an exclusive list. Only one color may be selected. Once a color is selected, selecting any other color will deselect the first color.

Exclusive lists may (and by default do) appear without any item initially selected. This may lead to problems if the application tries to apply any changes before an item is selected (such as within a delayed-mode dialog box). If your application needs an item to be selected at all times in order to operate correctly, your list should appear with one of the items initially selected (or with the "apply" trigger disabled).

Exclusive-None Lists

Exclusive-None lists are designed for cases in which either one item must be selected or no items should be selected. For example, a word processor may have a "Justifications" menu with several options such as left-justified, right-justified, or center-justified. Selecting any of these options will perform the justification operation on the current paragraph, but the user may also deselect any option, leaving no justification on the paragraph.

Exclusive-None lists by default appear without any items initially selected. Your application should, of course, be able to handle this case of nothing selected; if it cannot, the list should probably not be exclusive-none.

Extended-Selection Lists

Extended-Selection lists are designed for cases in which the application usually needs an exclusive selection but also desires to let the user select multiple items for bulk operations. Multiple selections are stored as lists of items.

For example, a disk copy operation may have an extended-selection list to specify the drive to copy a file to. In most cases, the user will only want to copy to one drive, and the list normally operates in exclusive mode to accommodate this. By extending the selection, however, the user can request the list to copy the file to multiple drives.

It is up to the specific UI how the user may extend a selection. In OSF/Motif, the user may select another item in tandem with a special keystroke (the SHIFT key) to extend the selection from one item to several. Extending a selection to an already selected item will deselect that item, leaving any other items unchanged.

Non-Exclusive Lists

Non-exclusive lists are designed for cases in which any or all of the items may be selected at once. Multiple selections, as in the extended-selection GenItemGroup, are stored as lists of items. Selecting an already selected item will deselect that item. Therefore, there may be a state where nothing is selected within a non-exclusive list.

For example, an interactive tutorial may ask the user if he wants more information on certain topics, giving these topics in a non-exclusive list. The user can freely choose as many or as few items as he wishes, and then select “Print” to print out information on those items.

In most cases, however, you might rather use a GenBooleanGroup object in place of a non-exclusive GenItemGroup. A GenBooleanGroup acts similar to a non-exclusive list but is streamlined to operate efficiently for lists with 16 or fewer items. (GenBooleanGroups represent their children's state through a word-length bitfield; therefore they may only accommodate 16 children.) You may also need to use a GenBooleanGroup if you need to keep track of individual indeterminate and modified states.

MSG_GEN_ITEM_GROUP_GET_BEHAVIOR_TYPE returns the behavior type stored in the list's *GIGI_behaviorType* instance field. To set a new behavior type, send the list a MSG_GEN_ITEM_GROUP_SET_BEHAVIOR_TYPE.

■ MSG_GEN_ITEM_GROUP_GET_BEHAVIOR_TYPE

```
GenItemGroupBehaviorType MSG_GEN_ITEM_GROUP_GET_BEHAVIOR_TYPE();
```

This message returns the current behavior type of a GenItemGroup (stored in the *GIGI_behaviorType* instance field).

Source: Unrestricted.

Destination: Any GenItemGroup.

Parameters: Nothing.

Return: The **GenItemGroupBehaviorType** of the GenItemGroup.

Interception: Generally not intercepted.

■ MSG_GEN_ITEM_GROUP_SET_BEHAVIOR_TYPE

```
void      MSG_GEN_ITEM_GROUP_SET_BEHAVIOR_TYPE(
          GenItemGroupBehaviorType   behaviorType);
```

This message sets a new behavior type for a GenItemGroup by changing the object's *GIGI_behaviorType* instance field. This message should only be sent while the object is not usable.

Source: Unrestricted.

Destination: Any non-usable GenItemGroup object.

Parameters: *behaviorType* The new **GenItemGroupBehaviorType**.

Return: Nothing.

Interception: Generally not intercepted.

11.3.3.2 GenItemGroup Selections

```
GIGI_selection, GII_identifier, GIGI_numSelections,
MSG_GEN_ITEM_GROUP_SET_NONE_SELECTED,
MSG_GEN_ITEM_GROUP_SET_SINGLE_SELECTION,
MSG_GEN_ITEM_GROUP_SET_MULTIPLE_SELECTIONS,
MSG_GEN_ITEM_GROUP_GET_SELECTION,
MSG_GEN_ITEM_GROUP_GET_NUM_SELECTIONS,
MSG_GEN_ITEM_GROUP_GET_MULTIPLE_SELECTIONS
```

Within a GenItemGroup, each GenItem must contain an identifier, often an enumerated value, to uniquely identify that item. The GenItemGroup will use this identifying value whenever referencing the GenItem (or items), either through its instance data or through messages. This value is stored in the GenItem's *GII_identifier* instance data.

In most cases, your collection of GenItems within a particular GenItemGroup should be different enumerated values. Each item within a specific GenItemGroup must have a unique identifier to reference each item. To ensure this, it is usually convenient to use an enumerated type. Make sure that you don't use two different enumerated types because they may not contain unique values across types.

In an exclusive or exclusive-none list, the GenItemGroup will store the identifier of the current selection in the *GIGI_selection* instance field. If no item is selected, this instance field will instead store the value GIGS_NONE (-1). (For an exclusive list, this case could only arise before an item is initially selected.)

Note that -1 is still a legal identifier for a GenItem. If you do use this identifier you must be sure to check the *GIGI_numSelections* instance field to check whether the *GIGI_selection* value corresponds to an item or to the GIGS_NONE value.

GenItemGroups by default appear with no items initially set (*GIGI_selection* is set to GIGS_NONE). If you wish a GenItemGroup to initially appear with a selection set to one of the GenItems, set *GIGI_selection* to the identifier of that GenItem.

Code Display 11-3 Setting Enumerated Types As Identifiers

```
/* In most cases, each GenItemGroup should have a single set of enumerated types
 * to name its children. In this case, MyCityGroup will have settings all related
 * to the enumerated type USCities, and San Francisco will be the initial
 * selection. */

typedef ByteEnum USCitiesEnum;
#define CITY_CHICAGO 1
#define CITY_NEW_YORK 2
#define CITY_SAN_FRANCISCO 3
#define CITY_LOS_ANGELES 4

@object GenItemGroupClass MyCityGroup = {
    GI_comp = @Chicago, @NewYork, @SanFrancisco, @LosAngeles;
    GIGI_selection = CITY_SAN_FRANCISCO;
}

@object GenItemClass Chicago = {
    GI_visMoniker = ‘C’, “Chicago”;
    GII_identifier = CITY_CHICAGO;
}

@object GenItemClass NewYork = {
    GI_visMoniker = ‘N’, “New York”;
    GII_identifier = CITY_NEW_YORK;
}

@object GenItemClass SanFrancisco = {
    GI_visMoniker = ‘S’, “San Francisco”;
    GII_identifier = CITY_SAN_FRANCISCO;
}

@object GenItemClass LosAngeles = {
    GI_visMoniker = ‘L’, “Los Angeles”;
    GII_identifier = CITY_LOS_ANGELES;
}
```

In an extended-selection or non-exclusive list, more than one item may be selected at a time. In this case, the *GIGI_selection* instance field will not store a single identifier but instead will store a ChunkHandle to a list of identifiers. This functionality is internal to the operation of the list.

You may set up a list (if it is extended-selection or non-exclusive) to initially appear with multiple selections. The list of identifiers may be of arbitrary size but must be manually set up in its own chunk. Set *GIGI_selection* to the ChunkHandle of the chunk containing this list. In this case, you must also set *GIGI_numSelections* to the proper number of selections. (See Code Display 11-4.)

For multiple selection lists (GIGBT_EXTENDED_SELECTION or GIGBT_NON_EXCLUSIVE) *GIGI_numSelections* may be set greater than 1 if more than one item is selected. In this case, *GIGI_selection* will contain a ChunkHandle to a list of identifiers instead of a single identifier.

Code Display 11-4 Setting Multiple Initial Selections

```
/* To set a GenItemGroup to appear initially with multiple selections, set up a
 * chunk containing the identifiers and a ChunkHandle in the GIGI_selection
 * instance field. Only the new list chunk and the GenItemGroup are shown; other
 * information is in the previous code display. */

@object GenItemGroupClass MyCityGroup = {
    GI_comp = @Chicago, @NewYork, @SanFrancisco, @LosAngeles;
    GIGI_behaviorType = GIGBT_NON_EXCLUSIVE;
    GIGI_numSelections = 2;
    GIGI_selection = (ChunkHandle) ChunkOf(@SelectionList);
}

@chunk word SelectionList[2] = {
    CITY_CHICAGO, CITY_SAN_FRANCISCO
};
```

In most cases, the user will change the selections of the GenItemGroup by clicking on items. Your application may also retrieve or alter the items selected with the following messages. In many of these messages, you may have to pass an indeterminate state; if you are unsure about what indeterminate state a list object should be in, pass FALSE. Most GenItemGroups do not need to worry about their indeterminate state.

None of the messages which alter the state or number of selections cause an apply or status message to be sent out by the GenItemGroup. If you wish to send out an apply or a status message in these cases, mark the GenItemGroup modified and send the object MSG_GEN_APPLY or MSG_GEN_ITEM_GROUP_SEND_STATUS_MSG.

MSG_GEN_ITEM_GROUP_GET_SELECTION returns the currently selected item for the GenItemGroup. If there are no items currently selected, this message will return GIGS_NONE. If there is more than one item selected, this message will only return the first item selected in that GenItemGroup. (You should use MSG_GEN_ITEM_GROUP_GET_MULTIPLE_SELECTIONS in this case.)

MSG_GEN_ITEM_GROUP_GET_MULTIPLE_SELECTIONS returns the current list of selections for non-exclusive and extended-selection lists. You must pass this message a pointer to a buffer to hold the list of selections, along with the number of items the buffer can hold. Make sure to allocate enough space in this buffer, or the buffer will not be filled in. Also make sure to send this message with **@call** and not **@send**, as it passes a pointer. You can check for the number of items currently selected with MSG_GEN_ITEM_GROUP_GET_NUM_SELECTIONS.

MSG_GEN_ITEM_GROUP_SET_NONE_SELECTED deselects any settings currently on so that the list will appear with no items selected. It takes a flag indicating whether the list should be in its indeterminate state. This message will also clear the object's modified flag. If you wish to set this flag, send the GenItemGroup a MSG_GEN_ITEM_GROUP_SET_MODIFIED_STATE after sending this message.

MSG_GEN_ITEM_GROUP_SET_SINGLE_SELECTION sets a single selection for a GenItemGroup. Any old selections will be deselected automatically, regardless of the behavior type of the list. It takes a flag indicating whether the list should be in its indeterminate state. If the GenItemGroup is scrollable and exclusive, this message will usually ensure that the list will automatically scroll to the selected item. This message also clears the object's modified flag.

MSG_GEN_ITEM_GROUP_SET_MULTIPLE_SELECTIONS sets a a group of selections for a GenItemGroup. Any previously selected items will be deselected. The message must pass the number of selections and a pointer to the list of identifiers to set. This message will clear the GenItemGroup's modified flag.

■ MSG_GEN_ITEM_GROUP_GET_SELECTION

```
word      MSG_GEN_ITEM_GROUP_GET_SELECTION();
```

This message returns the current selection identifier for the GenItemGroup. If there is no selection, this message returns GIGS_NONE. If there are multiple selections, this message only returns the first selection. Use MSG_GEN_ITEM_GROUP_GET_MULTIPLE_SELECTIONS for multiple selections. A selection may be returned even if the item is not usable, not enabled, or not in the GenItemGroup.

If you are using a GenItemGroup which may contain a selection of -1, you will want to call MSG_GEN_ITEM_GROUP_GET_NUM_SELECTIONS to differentiate between the item with -1 being selected and GIGS_NONE.

Source: Unrestricted.

Destination: Any GenItemGroup object.

Return: The current selection stored in the GenItemGroup's *GIGI_selection* instance field (or GIGS_NONE if there are no selections).

Interception: Generally not intercepted.

■ MSG_GEN_ITEM_GROUP_GET_MULTIPLE_SELECTIONS

```
word      MSG_GEN_ITEM_GROUP_GET_MULTIPLE_SELECTIONS(
          word   *selectionList,
          word   maxSelections);
```

This message returns the current list of selections for non-exclusive and extended-selection GenItemGroups. The caller must allocate a buffer for the entries and pass the size of that buffer. If there is insufficient space in the passed buffer, no entries will be filled in. You should call MSG_GEN_ITEM_GET_NUM_SELECTIONS beforehand in order to allocate the correct maximum size for the buffer.

This message returns the number of selections filled into the buffer. The buffer is filled in with the proper identifiers. The list of identifiers within the buffer will not be null-terminated.

If you are using an exclusive or exclusive-none list, use MSG_GEN_ITEM_GROUP_GET_SELECTION instead.

Source: Unrestricted.

Destination: Any GenItemGroup object.

Parameters: *selectionList* A pointer to the buffer to store the selection entries.

maxSelections The maximum number of selections that may be returned by the message.

Return: The total number of selections returned in the buffer.

selectionList The pointer to the buffer containing the identifiers of the selections.

Interception: Generally not intercepted.

■ MSG_GEN_ITEM_GROUP_GET_NUM_SELECTIONS

```
word       MSG_GEN_ITEM_GROUP_GET_NUM_SELECTIONS();
```

This message returns the GenItemGroup's current number of selections (stored in *GIGI_numSelections*), or zero if there is no selection.

Source: Unrestricted.

Destination: Any GenItemGroup object.

Return: The number of selections in the GenItemGroup.

Interception: Generally not intercepted.

■ MSG_GEN_ITEM_GROUP_SET_NONE_SELECTED

```
void       MSG_GEN_ITEM_GROUP_SET_NONE_SELECTED(
           Boolean             indeterminate);
```

This message sets a GenItemGroup to show no selections; it should not be sent to an exclusive GenItemGroup. This message will also clear the GenItemGroup's modified flag. If you wish to set the modified flag, send the list a MSG_GEN_ITEM_GROUP_SET_MODIFIED_STATE.

Source: Unrestricted.

Destination: Any GenItemGroup object.

Parameters: *indeterminate* Pass TRUE if the GenItemGroup should be marked indeterminate, FALSE otherwise.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_ITEM_GROUP_SET_SINGLE_SELECTION

```
void      MSG_GEN_ITEM_GROUP_SET_SINGLE_SELECTION(
          word               identifier,
          Boolean            indeterminate);
```

This message sets exactly one selection for a GenItemGroup. Any previously-selected items will be deselected, regardless of the GenItemGroup's behavior type. To set items without disturbing other settings, use MSG_GEN_ITEM_GROUP_SET_ITEM_STATE. If the list is on-screen, this message will cause a visual update. If the GenItemGroup is scrollable, this message will usually ensure that the item selected will appear on-screen.

This message clears the list's modified state. If you wish to set the group modified, send it a MSG_GEN_ITEM_GROUP_SET_MODIFIED_STATE. You may set a selection that is not usable, not enabled, or even not in the GenItemGroup. (This is useful for linked GenItemGroups.)

Source: Unrestricted.

Destination: Any GenItemGroup.

Parameters: *identifier* The identifier (enumerated value) of the item to select.

indeterminate Pass TRUE if the group should be marked indeterminate.

Return: Nothing.

Interception: Generally not intercepted.

Tips: An item with a matching identifier need not be present to become the GenItemGroup's "selection." Using this knowledge, you can set up several "linked" GenItemGroups, each with only one selection. (See "GenItemGroup Links" on page 752 for information and examples.)

■ MSG_GEN_ITEM_GROUP_SET_MULTIPLE_SELECTIONS

```
void      MSG_GEN_ITEM_GROUP_SET_MULTIPLE_SELECTIONS(
          word   *selectionList,
          word   numSelections);
```

This message sets multiple selections for a GenItemGroup. The caller must pass a pointer to a list of identifiers that should be selected, along with the number of selections to set. This message will clear the GenItemGroup's modified flag. If you wish to set the group modified send it a MSG_GEN_ITEM_GROUP_SET_MODIFIED_STATE.

Source: Unrestricted.

Destination: Any GenItemGroup object.

Parameters: *selectionList* A pointer to the buffer containing a list of item identifiers.

numSelections The number of selections in the passed list.

Return: Nothing.

Interception: Generally not intercepted.

11.3.3.3 GenItemGroup States

```
GIGI_stateFlags,
ATTR_GEN_ITEM_GROUP_SET_MODIFIED_ON_REDUNDANT_SELECTION,
MSG_GEN_ITEM_GROUP_SET_INDETERMINATE_STATE,
MSG_GEN_ITEM_GROUP_IS_INDETERMINATE,
MSG_GEN_ITEM_GROUP_SET_MODIFIED_STATE,
MSG_GEN_ITEM_GROUP_IS_MODIFIED
```

The *GIGI_stateFlags* contains flags relating to the state of the entire GenItemGroup. The two flags possible are

- GIGSF_INDETERMINATE
  This flag marks the GenItemGroup indeterminate. Only the group may be indeterminate; individual GenItems may not be marked indeterminate.

- GIGSF_MODIFIED
  This flag marks the GenItemGroup modified. Only the group may be marked as modified; individual GenItems may not be marked as modified.

A GenItemGroup is in the indeterminate state if its *GIGI_stateFlags* contains the GIGSF_INDETERMINATE flag. GenItems cannot be in an indeterminate state by themselves; either the entire list is indeterminate or it is not. If you want individual items to be indeterminate, you may want to use a GenBooleanGroup object instead.

A GenItemGroup is in the modified state if its *GIGI_stateFlags* contains the GIGSF_MODIFIED flag. GenItems may not be in a modified state by themselves; either the entire list is modified or it is not. If you want to mark individual items as modified, you may want to use a GenBooleanGroup object instead.

In immediate mode, any user change to the GenItemGroup's state automatically sets the GIGSF_MODIFIED flag and also forces an "apply." Therefore, in immediate mode, any user changes will, by default, cause the apply action to be sent out immediately. In delayed mode, the GIGSF_MODIFIED flag is set, but an apply action will not automatically occur. The apply action will only occur when the user forces it (such as clicking on an "apply" trigger) or when the application forces it (by sending out a MSG_GEN_APPLY).

You can force an apply by sending the GenItemGroup MSG_GEN_APPLY. You can also override the default behavior, forcing the object to send out its notification message on all applies, by including ATTR_GEN_SEND_APPLY_MSG_ON_APPLY_EVEN_IF_NOT_MODIFIED in the object's instance data. Any time the object receives an "apply," the GenItemGroup will send out the notification message even if the object does not contain any modified GenItem children.

In general, if the user makes redundant changes (such as repeatedly selecting the same item) the group will not be marked modified—the group has not changed state since the last apply, so there is no need to send out a notification message. If, however, you wish for the group to be set modified on redundant changes, add the vardata attribute ATTR_GEN_ITEM_GROUP_SET_MODIFIED_ON_REDUNDANT_SELECTION to the object's instance data. Whenever the user makes any selection, the group will be marked modified.

To check whether a GenItemGroup is indeterminate, send it a MSG_GEN_ITEM_GROUP_IS_INDETERMINATE. The message will return *true* if the list is indeterminate, *false* if it is not.

To set a GenItemGroup indeterminate, send it MSG_GEN_ITEM_GROUP_SET_INDETERMINATE_STATE. Note that the messages to set a GenItemGroup's selections already provide an argument to set the GenItemGroup indeterminate.

To check whether a GenItemGroup has been modified, send it MSG_GEN_ITEM_GROUP_IS_MODIFIED. This will return *true* if the list has been modified, *false* if it has not. To set a GenItemGroup modified, send it a MSG_GEN_ITEM_GROUP_SET_MODIFIED_STATE.

■ MSG_GEN_ITEM_GROUP_SET_INDETERMINATE_STATE

```
void     MSG_GEN_ITEM_GROUP_SET_INDETERMINATE_STATE(
         Boolean              indeterminateState);
```

This message sets the GenItemGroup's GIGSF_INDETERMINATE flag in its *GIGI_stateFlags* instance field. Usually this message is used in conjunction with MSG_GEN_ITEM_GROUP_SET_MULTIPLE_SELECTIONS. (MSG_GEN_ITEM_GROUP_SET_NONE_SELECTED and MSG_GEN_ITEM_GROUP_SET_SINGLE_SELECTION set the indeterminate state themselves.)

Source: Unrestricted.

Destination: Any GenItemGroup object.

Parameters: *indeterminate* Pass TRUE to mark the group indeterminate, FALSE to set the group not indeterminate.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_ITEM_GROUP_IS_INDETERMINATE

```
Boolean  MSG_GEN_ITEM_GROUP_IS_INDETERMINATE();
```

This message checks whether a GenItemGroup is in the indeterminate state; it checks the GIGSF_INDETERMINATE flag in the GenItemGroup's *GIGI_stateFlags*.

Source: Unrestricted.

Destination: Any GenItemGroup object.

Return: TRUE if the GenItemGroup is indeterminate, FALSE otherwise.

Interception: Generally not intercepted.

■ MSG_GEN_ITEM_GROUP_SET_MODIFIED_STATE

```
void      MSG_GEN_ITEM_GROUP_SET_MODIFIED_STATE(
          Boolean             modifiedState);
```

This message marks a GenItemGroup modified by setting the GIGSF_MODIFIED flag in its *GIGI_stateFlags* instance field.

Source: Unrestricted.

Destination: Any GenItemGroup object.

Parameters: *modifiedState* TRUE to mark the list modified, FALSE otherwise.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_ITEM_GROUP_IS_MODIFIED

```
Boolean   MSG_GEN_ITEM_GROUP_IS_MODIFIED();
```

This message checks whether a GenItemGroup has been modified by checking the GIGSF_MODIFIED flag in the GenItemGroup's *GIGI_stateFlags*.

Source: Unrestricted.

Destination: Any GenItemGroup object.

Parameters: None.

Return: TRUE if GenItemGroup is marked modified, FALSE otherwise.

Interception: Generally not intercepted.

11.3.3.4 Sending the Event

```
GIGI_applyMsg, GIGI_destination, GEN_ITEM_GROUP_APPLY_MSG,
ATTR_GEN_SEND_APPLY_MSG_ON_APPLY_EVEN_IF_NOT_MODIFIED,
MSG_GEN_ITEM_GROUP_GET_DESTINATION,
MSG_GEN_ITEM_GROUP_SET_DESTINATION,
MSG_GEN_ITEM_GROUP_GET_APPLY_MSG,
MSG_GEN_ITEM_GROUP_SET_APPLY_MSG
```

When the GenItemGroup receives notice to apply any changes to its state, it will first check its GIGSF_MODIFIED flag. If the list has been modified, the GenItemGroup will send the apply action stored in its *GIGI_applyMsg* field to the destination stored in its *GIGI_destination* field.

GIGI_applyMsg stores the apply action you wish the GenItemGroup to send out whenever you make a selection choice within the group. This message should be based on the prototype GEN_ITEM_GROUP_APPLY_MSG so that the proper arguments (*selection*, *numSelections*, and *stateFlags*) are automatically passed. *GIGI_destination* should store the object or process you wish to handle the message sent out by this group. This object should have a handler for the *GIGI_applyMsg*.

You can force a GenItemGroup to apply its changes with MSG_GEN_APPLY. You can also override the default behavior, forcing the object to send out its apply action on all applies, whether modified or not, by including ATTR_GEN_SEND_APPLY_MSG_ON_APPLY_EVEN_IF_NOT_MODIFIED in the object's instance data. Any time the object needs to apply its changes, the GenItemGroup will send out the notification message even if the object is not marked GIGSF_MODIFIED. 11.3

To retrieve the object or process stored in the GenItemGroup's *GIGI_destination* instance field or the message stored in its *GIGI_applyMsg* instance field, send the list MSG_GEN_ITEM_GROUP_GET_DESTINATION or MSG_GEN_ITEM_GROUP_GET_APPLY_MSG.

To set the GenItemGroup's destination, send the list MSG_GEN_ITEM_GROUP_SET_DESTINATION, passing the optr of the new destination. To set the GenItemGroup's notification message, send the list MSG_GEN_ITEM_GROUP_SET_APPLY_MSG, passing it the message to use.

■ GEN_ITEM_GROUP_APPLY_MSG

```
void      GEN_ITEM_GROUP_APPLY_MSG(
          word   selection,
          word   numSelections
          byte   stateFlags);
```

Use this prototype to define the apply message for your GenItemGroup (stored in the *GIGI_applyMsg* instance field). This prototype ensures that the message passes the correct parameters to your message handler.

Source: Your GenItemGroup object.

Destination: Destination of your GenItemGroup.

Parameters:

selection	The current selection of the GenItemGroup.
numSelections	The current number of selections of the GenItemGroup.

stateFlags The current state flags of the GenItemGroup.

Return: Nothing.

■ MSG_GEN_ITEM_GROUP_GET_DESTINATION

```
optr    MSG_GEN_ITEM_GROUP_GET_DESTINATION();
```

This message returns the optr of the current destination object as specified in the *GIGI_destination* instance field.

Source: Unrestricted.

Destination: Any GenItemGroup object.

Parameters: Nothing.

Return: The optr of the current destination object.

Interception: Generally not intercepted.

■ MSG_GEN_ITEM_GROUP_SET_DESTINATION

```
void    MSG_GEN_ITEM_GROUP_SET_DESTINATION(
        optr   dest);
```

This message sets a new destination object for the list, stored in the list's *GIGI_destination* field. The apply and status messages of this GenItemGroup will be sent to this new destination.

Source: Unrestricted.

Destination: Any GenItemGroup.

Parameters: *dest* The optr of the new destination object to be put in *GIGI_destination*.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_ITEM_GROUP_GET_APPLY_MSG

```
Message MSG_GEN_ITEM_GROUP_GET_APPLY_MSG();
```

This message returns the current apply message in the GenItemGroup's *GIGI_applyMsg* instance field.

Source: Unrestricted.

Destination: Any GenItemGroup.

Parameters: Nothing.

Return: The current apply message.

Interception: Generally not intercepted.

■ MSG_GEN_ITEM_GROUP_SET_APPLY_MSG

```
void    MSG_GEN_ITEM_GROUP_SET_APPLY_MSG(
        Message             message);
```

This message sets a new apply message in the GenItemGroup's *GIGI_applyMsg* instance field.

Source: Unrestricted.

Destination: Any GenItemGroup object.

Parameters: *message* The new notification message.

Return: Nothing.

Interception: Generally not intercepted.

11.3.3.5 Sending Status Messages

GEN_ITEM_GROUP_STATUS_MSG, ATTR_GEN_ITEM_GROUP_STATUS_MSG, MSG_GEN_ITEM_GROUP_SEND_STATUS_MSG

If you wish your application to receive notice whenever the list's state changes, even if those changes will not be immediately applied, you may set a status message in the object's instance data.

For example, a color selection dialog box provides two means to set the RGB value of a color: one in a GenItemGroup listing the total possible states and another in a GenBooleanGroup with separate toggle states for Red, Green, and Blue. In delayed mode, setting the values in one group should change the display of the other group simultaneously but should not result in an apply action until the user activates the apply trigger.

One list can notify its partner of state changes with a status message. The other list can respond to state changes (and therefore change its corresponding state) without actually applying those changes.

To set up a status message, add a message based on the prototype GEN_ITEM_GROUP_STATUS_MSG. This message will be sent to the list's destination object. Set ATTR_GEN_ITEM_GROUP_STATUS_MSG to the status message.

Code Display 11-5 Status Message in GenItemGroup

```
/* Use the prototype GEN_ITEM_GROUP_STATUS_MSG to set up a status message. The
 * prototype takes three arguments: selection (the contents of GIGI_selection),
 * numSelections (the contents of GIGI_numSelections), and stateFlags (the contents
 * of GIGI_stateFlags). */

        /* In the destination's class definition: */
@message (GEN_ITEM_GROUP_STATUS_MSG) MSG_COLOR_STATUS_CHANGE;

        /* In the GenItemGroup's object declaration, set
         * ATTR_GEN_ITEM_GROUP_STATUS_MSG to the message you declared. */
@object GenItemGroupClass MyColorGroup = {
    GIGI_applyMsg = MSG_COLOR_CHANGE;
    GIGI_destination = process;
    ATTR_GEN_ITEM_GROUP_STATUS_MSG = MSG_COLOR_STATUS_CHANGE;
}

        /* Finally, in your handler for this status message, do required work. */
@method MyProcessClass, MSG_COLOR_STATUS_CHANGE {
    switch (selection) {
        case C_BLACK:
            /* ...(code)... */
        case C_WHITE:
            /* ...(code)... */
    }
}
```

■ GEN_ITEM_GROUP_STATUS_MSG

```
void      GEN_ITEM_GROUP_STATUS_MSG(
          word   selection,
          word   numSelections,
          byte   stateFlags);
```

Use this prototype to define a status message for your GenItemGroup (stored in the ATTR_GEN_ITEM_GROUP_STATUS_MSG vardata). This prototype ensures that your status message passes the correct parameters.

Source: Your GenItemGroup object.

Destination: The destination of your GenItemGroup.

Parameters: *selection* The current selection (enumerated type stored in *GIGI_selection*) of the GenItemGroup.

numSelections The current number of selections (stored in *GIGI_numSelections*) of the GenItemGroup.

stateFlags The current state flags (stored in *GIGI_stateFlags*) of the GenItemGroup.

Return: Nothing.

Tips: It is usually convenient to set up a switch statement based on the enumerated value of the *selection* passed in this message handler.

■ MSG_GEN_ITEM_GROUP_SEND_STATUS_MSG

```
void    MSG_GEN_ITEM_GROUP_SEND_STATUS_MSG(
        Boolean modifiedState);
```

This message causes the GenItemGroup to send out the status message to the destination object. This status message is stored in ATTR_GEN_ITEM_GROUP_STATUS_MSG.

Source: Unrestricted.

Destination: Any GenItemGroup object that contains a status message.

Parameters: *modifiedState* Pass TRUE if the GenItemGroup should be marked modified.

Return: Nothing.

Interception: Generally not intercepted.

11.3.4 Working with Items

With several GenItem and GenItemGroup messages, you can work with individual items. The following sections detail these messages.

11.3.4.1 Altering the Identifiers

MSG_GEN_ITEM_GET_IDENTIFIER, MSG_GEN_ITEM_SET_IDENTIFIER, MSG_GEN_ITEM_GROUP_GET_UNIQUE_IDENTIFIER

You may change the identifiers of GenItems dynamically. To retrieve a GenItem's identifier, send the GenItem MSG_GEN_ITEM_GET_IDENTIFIER.

To set a GenItem's identifier, send it MSG_GEN_ITEM_SET_IDENTIFIER, passing the message the value you wish to set it to. Make sure that this value is unique and *not* duplicated by any other GenItem in the same GenItemGroup.

To generate a unique identifier for an item in a group, the GenItemGroup may use MSG_GEN_ITEM_GROUP_GET_UNIQUE_IDENTIFIER.

11.3

■ MSG_GEN_ITEM_GET_IDENTIFIER

```
word      MSG_GEN_ITEM_GET_IDENTIFIER();
```

This message returns the identifier stored in the item's *GII_identifier* instance field.

Source: Unrestricted.

Destination: Any GenItem object.

Return: The identifier of the GenItem object.

Interception: Generally not intercepted.

■ MSG_GEN_ITEM_SET_IDENTIFIER

```
void      MSG_GEN_ITEM_SET_IDENTIFIER(
          word   identifier);
```

This message sets a GenItem's *GII_identifier* instance field to a new value. There is no effect on the list itself. The item whose identifier is changed must not be currently selected. It is therefore up to the application to make sure that the item is de-selected before changing its identifier.

Source: Unrestricted.

Destination: Any GenItem object.

Return: Nothing.

Interception: Generally not intercepted.

Warnings: The new enumerated value should be unique among all of the items in the list. Use MSG_GEN_ITEM_GROUP_GET_UNIQUE_IDENTIFIER.

■ MSG_GEN_ITEM_GROUP_GET_UNIQUE_IDENTIFIER

```
word      MSG_GEN_ITEM_GROUP_GET_UNIQUE_IDENTIFIER();
```

This message returns an identifier that is unique among the items in the GenItemGroup's group. It does not check across links, however.

◆ Objects

Source: Unrestricted.

Destination: Any non-usable GenItemGroup object.

Parameters: None.

Return: The unique item identifier.

Interception: Generally not intercepted.

11.3.4.2 Altering the Item's State

```
MSG_GEN_ITEM_GROUP_GET_ITEM_OPTR,
MSG_GEN_ITEM_GROUP_SET_ITEM_STATE,
MSG_GEN_ITEM_GROUP_IS_ITEM_SELECTED
```

You can retrieve the optr of a selection by sending the GenItemGroup a MSG_GEN_ITEM_GROUP_GET_ITEM_OPTR, passing the identifier of the requested item. You may then use this item optr to send messages directly to the individual items or to set items enabled/disabled, usable/not usable, etc. If the item is not found, this message will return null.

You may also set an item selected without altering any other settings in the GenItemGroup. Send the list a MSG_GEN_ITEM_GROUP_SET_ITEM_STATE, passing it the identifier of the item to mark as selected. If the list is exclusive or exclusive-none, any other settings will be deselected; if the list is extended-selection or non-exclusive, the other settings will remain intact.

To check on whether an item is selected, send the GenItemGroup a MSG_GEN_ITEM_GROUP_IS_ITEM_SELECTED, passing it the identifier of the item in question. This message will return *true* if the item is currently selected, *false* if it is not.

Send MSG_GEN_ITEM_GROUP_MAKE_ITEM_VISIBILE to force a scrolling GenItemGroup to scroll to an item. Pass this message the identifier of the requested item. This message has no effect on non-scrolling lists.

■ MSG_GEN_ITEM_GROUP_GET_ITEM_OPTR

```
optr    MSG_GEN_ITEM_GROUP_GET_ITEM_OPTR(
        word   identifier);
```

This message returns the optr of a GenItem within a GenItemGroup. You may then use this optr for directly manipulating the GenItem. If an item with the requested identifier is not found as one of the GenItemGroup's children, this message will return a null optr.

Source: Unrestricted.

Destination: Any GenItemGroup object.

Parameters: *identifier* The identifier of the item to search for (stored in the GenItem's *GII_identifier* instance field).

Return: The optr of the GenItem with the requested identifier (or a null optr if none is found).

Interception: Generally not intercepted.

■ MSG_GEN_ITEM_GROUP_SET_ITEM_STATE

```
void        MSG_GEN_ITEM_GROUP_SET_ITEM_STATE(
            word                identifier,
            Boolean             state);
```

This message sets the state (selected or unselected) of a particular GenItem within a GenItemGroup. Other items will be unaffected by the new state. This message also clears the GenItemGroup's modified state. If an identifier is passed which does not match an item within the GenItemGroup (or matches a not usable item), it will still be made the selection.

Source: Unrestricted.

Destination: Any GenItemGroup object.

Parameters: *identifier* The identifier of the item to change state.

state TRUE to mark item selected, FALSE to mark item unselected.

Return: Nothing.

Interception: Generally not intercepted.

Tips: An item with a matching identifier need not be present to become the GenItemGroup's selection. Using this knowledge, you can set up several "linked" GenItemGroups, each with only one selection. (See "GenItemGroup Links" on page 752 for information and examples.)

■ MSG_GEN_ITEM_GROUP_IS_ITEM_SELECTED

```
Boolean MSG_GEN_ITEM_GROUP_IS_ITEM_SELECTED(
        word   identifier);
```

This message checks whether an item within a GenItemGroup is selected, even if that item is not usable, not enabled, or not within the GenItemGroup. The message returns *true* if the item is selected, *false* if it is not selected.

Source: Unrestricted.

Destination: Any GenItemGroup object.

Parameters: *identifier* The identifier of the item to check for.

Return: Will return *true* if the item is selected, *false* if it is not.

Interception: Generally not intercepted.

11.3.5 Scrolling GenItemGroups

```
HINT_ITEM_GROUP_SCROLLABLE,
MSG_GEN_ITEM_GROUP_MAKE_ITEM_VISIBLE
```

A scrolling list is a special version of a GenItemGroup that allows the user to scroll through a list of items which are not all visible at one time. Scrolling lists usually provide scrollbars or some other means to quickly scan through a list. You may make a GenItemGroup scrollable if the specific UI allows by including the hint HINT_ITEM_GROUP_SCROLLABLE in the list's object declaration.

The list will come up in a standard size determined by the specific UI. You may override this by adding a size hint to the object (HINT_FIXED_SIZE, HINT_INITIAL_SIZE, etc.). Notification of changes for a scrolling list, as well as setting and getting the selections, works the same for a scrolling list as for a normal GenItemGroup. If an item is selected, the scrolling list will almost always scroll automatically to the selected item.

To force a GenItemGroup to scroll to an item, send MSG_GEN_ITEM_GROUP_MAKE_ITEM_VISIBLE. Pass this message the identifier of the item to be made visible in the scrolling list. This message has no effect on a non-scrolling list.

You may wish in some cases for a scrolling list to be for display only. If so, set the scrollable GenItemGroup's *GI_attrs* instance data to include the flag GA_READ_ONLY. The user will be unable to interact with a display-only scrolling list (although, of course, the user will still be able to scroll through the list).

If your scrolling list will have many settings or will often be dynamically updated, you should consider using a GenDynamicList instead.

11.3

■ MSG_GEN_ITEM_GROUP_MAKE_ITEM_VISIBLE

```
void    MSG_GEN_ITEM_GROUP_MAKE_ITEM_VISIBLE(
        word   identifier);
```

This message ensures that an item within a scrolling GenItemGroup is visible, by scrolling the list if necessary. This message has no effect if the GenItemGroup is not scrollable or is not usable.

Source: Unrestricted.

Destination: Any scrollable, usable GenItemGroup object.

Parameters: *identifier* The identifier of the item to be made visible.

Return: Nothing.

Interception: Generally not intercepted.

11.3.6 GenItemGroup Links

ATTR_GEN_ITEM_GROUP_LINK

In some cases, geometry constraints interfere with functionality. For example, you may wish for "nested" GenItemGroup behavior. Each main selection should act as an exclusive selection within the GenItemGroup, but it should also allow other generic objects "in-between" these exclusive items. Normal geometry constraints prevent this, as GenItemGroups may only contain GenItems as children.

To enable this, you may set up each main section as a separate GenItemGroup with a single selection. Within each GenItemGroup, include a link to the next GenItemGroup with ATTR_GEN_ITEM_GROUP_LINK. Links should be circular; i.e., if three GenItemGroups should be linked, the first group should point to the second, the second to the third, and the third back to the first. The three linked GenItemGroups will act as one entity. If each GenItemGroup is exclusive, one group will select its single item child if selected and the other groups will deselect their children.

These links work because you are allowed to “select” items that are not actually within a GenItemGroup. If an item not within the group is selected, any current selection will therefore be de-selected.

11.3.7 Limitations of the GenItemGroup

GenItemGroups may only have GenItems as children. Likewise, GenItems must always be generic children of a GenItemGroup object. There also is no capability to allow some items under a GenItemGroup object to be exclusive while others are non-exclusive. This can often be accomplished instead by making the list non-exclusive and clearing other items when necessary to create the exclusive behavior.

In some cases, you may wish to store modified or indeterminate states for individual items rather than for only the whole group. In this case, you should probably use a GenBooleanGroup.

For long lists of items or lists that may require frequent updating, you might wish to use a GenDynamicList instead.

11.4 GenDynamicListClass

The **GenDynamicListClass** is a subclass of **GenItemGroupClass** that allows an application to pass monikers to the object as needed rather than specify a GenItem for each item in the list. In almost all cases, you will want a GenDynamicList to be a scrolling list so that not all of the list entries will be displayed at any given time. (You can, however, still choose to create a non-scrolling dynamic list.)

The GenDynamicList displays a list of monikers; this list of monikers is usually in the form of a data chunk managed by some other (application-supplied) object. For each moniker the GenDynamicList wishes to display, it will send out a query message and the position of the moniker requested. The object handling the query message will return a pointer to the moniker and will draw the moniker within the GenDynamicList. In this way, monikers for a GenDynamicList are only drawn as needed.

The dynamic list allows the following:

11.4

- Long Lists
  Very long lists can be stored by the application in any way it wants and as efficiently as possible. Because these lists are often scrolling lists, the UI is not cluttered up with a multitude of list entries. Only those within the scroller's bounds will be visually displayed.
- Dynamic Display of Data
  Applications can easily create a scrolling list from arbitrarily organized data. As long as you provide a means to convert the data into the proper moniker format, you can use any data you wish for moniker entries. You can also dynamically access this data and update the list entries.

An application specifies the number of items in the list and if desired can set up a custom size for the dynamic list. Whenever the object needs to display a moniker, it will query its destination for the visual moniker to display as an item in the list. At any time, the application can remove or add items from the dynamic list or change the total number of items and start with all new items. Because the list only queries for items as they become visible, it is more efficient than having to display a large number of entries in a long list.

Since **GenDynamicListClass** is a subclass of **GenItemGroupClass**, all of the same messages that are passed to GenItemGroups can be passed to GenDynamicList (except MSG_GEN_ITEM_GROUP_GET_ITEM_OPTR, which has no meaning for a GenDynamicList). The lists can be exclusive, exclusive-none, extended-selection, or non-exclusive. The user can select and deselect items depending on the behavior type, etc.

11.4.1 DynamicList Instance Data

GenDynamicListClass inherits all of the instance fields of **GenItemGroupClass**. The GenDynamicList object provides two additional instance fields:

```
@instance word GDLI_numItems = 0;
@instance word GDLI_queryMsg = 0;
```

GDLI_numItems stores the number of items in the dynamic list. This field should be set to the initial number of monikers needed within your GenDynamicList. If this value is initially unknown (i.e. is retrieved dynamically) then you should set this value after the number of items becomes known (using MSG_GEN_DYNAMIC_LIST_INITIALIZE). If this instance field is zero, no items will be drawn.

GDLI_queryMsg stores the querying message that the dynamic list sends each time it needs to display an entry's moniker. (For example, when initially realized, a scrolling dynamic list may display only four out of 16 entries; this message will be sent out four times to retrieve the monikers for the items initially displayable.)

The query message will be handled by whatever object you set in *GIGI_destination*. The handler should search a list of data for the visual monikers to display within the list. You will want to set up a custom handler for this message and the list of data to search through. Note that this list of data will serve as the GenDynamicList's "children"; there are no GenItems allowed as children of a GenDynamicList. The query message is based on the prototype GEN_DYNAMIC_LIST_QUERY_MSG.

■ GEN_DYNAMIC_LIST_QUERY_MSG

```
void      GEN_DYNAMIC_LIST_QUERY_MSG(
          optr   list,
          word   item);
```

Use this prototype to define the query message for your GenDynamicList (stored in *GDLI_queryMsg*). This prototype ensures that the message passes the correct parameters to your message handler.

Source: Your GenDynamicList object.

Destination: The destination of your GenDynamicList object (*GIGI_destination*).

Parameters:

list	The optr of the list.
item	The item (position number) needing a moniker.

Return: Nothing.

11.4.2 DynamicList Basics

MSG_GEN_DYNAMIC_LIST_REPLACE_ITEM_MONIKER,
MSG_GEN_DYNAMIC_LIST_REPLACE_ITEM_MONIKER_OPTR,
MSG_GEN_DYNAMIC_LIST_REPLACE_ITEM_TEXT

You will need to perform several preparatory steps in order to use a GenDynamicList object, even in its most basic form. The following steps are required for all dynamic lists.

- Create a GenDynamicList object. You must include the number of items to appear in your list (or set them dynamically when this number is known) and a query message to handle the extraction and display of monikers from a list of data.
- Create a list of data to use as entries in the list. For simplicity, this can be a chunk of data containing a list of visual monikers, although the data may come from any source—as long as you provide the means to convert the data into monikers.
- Create a query message handler to search through the list of data. This handler should convert any data into a proper format (if necessary) and send the moniker to the dynamic list to display the item. You can convert the data into a proper format by using one of several messages provided with **GenDynamicListClass**. These messages may take text strings, visual monikers, or other forms of data to set the monikers for the dynamic list.

Code Display 11-6 Creating a GenDynamicList object

```
/* In the definition of your destination object’s class, define a message based
 * on the prototype GEN_DYNAMIC_LIST_QUERY. This prototype passes two arguments:
 * list (the optr of the list sending info) and item (the selection enum). */

@message (GEN_DYNAMIC_LIST_QUERY_MSG) MSG_QUERY_COLOR_MONIKER;

/* Set up the selections, just as you would in a normal GenItemGroup. */

typedef enum {
    MC_RED, MC_YELLOW, MC_BLUE, MC_GREEN
} MyColors;
#define NUM_COLORS MC_GREEN+1
```

```
/* In the object declaration, include normal instance data for a GenItemGroup,
 * making sure to also fill in GDLI_numItems and GDLI_queryMsg with the proper
 * information. Note that there are no generic children provided for this object.
 * The list will retrieve and display monikers automatically through the query
 * message as they are needed. */

@object GenDynamicListClass MyList = {
    /* Set the initial selection MC_BLUE (and the number of items selected to 1).*/

    GIGI_selection = MC_BLUE;
    GIGI_applyMsg = MSG_NOTIFY_COLOR_CHANGE;
    GIGI_destination = process;

    /* Set numItems to the total number of items to appear in this list. If you
     * will not know this initially, make sure to set this instance data when you
     * do know (with MSG_GEN_DYNAMIC_LIST_INITIALIZE). */

    GDLI_numItems = NUM_COLORS;
    GDLI_queryMsg = MSG_QUERY_COLOR_MONIKER;
    HINT_ITEM_GROUP_SCROLLABLE;
}
```

When a dynamic list is first built, the specific UI will decide how many items within the scrolling list should be initially displayed. For each of these items, the dynamic list will send out a query message, passing the number (position) of the item to search for. The querying message will search through a list of data (either custom or pre-existing), extracting and converting the data into text, graphics strings, or visual monikers, etc.

Finally, your query message handler should set each moniker in the list using one of the MSG_GEN_DYNAMIC_LIST_REPLACE_ITEM_MONIKER messages. There are three messages to set the moniker of a dynamic list entry:

- MSG_GEN_DYNAMIC_LIST_REPLACE_ITEM_MONIKER
  This is a general purpose moniker replacement message. It can take a variety of source types (optrs, fptrs, hptrs), data types (visMonikers, text strings, graphics strings, or even geode tokens), and (in the case of graphics strings) the height and width of the moniker. This message also provides a flag allowing a moniker entry to appear visually disabled (grayed out in OSF/Motif).

- MSG_GEN_DYNAMIC_LIST_REPLACE_ITEM_MONIKER_OPTR
  This is a simplified version of the above message which takes a visual moniker and displays it within the dynamic list. The message's only

arguments are the item in the list to update and the optr of the moniker for that item.

- MSG_GEN_DYNAMIC_LIST_REPLACE_ITEM_TEXT
  This is another simplified version of the first message which takes a text string and displays it within the dynamic list. The message's only arguments are the item in the list to update and a pointer to the text string to use.

The simpler messages are useful if your data is a list of complete visual moniker structures or is a list of text strings. If you need to retrieve special data (such as GStrings or geode tokens), or if you need to perform special operations on the moniker (such as bringing it up disabled), then you must use MSG_GEN_DYNAMIC_LIST_REPLACE_ITEM_MONIKER.

The data itself may be placed in several areas of your code. If the data is simple text strings, set up within an array, it may appear within the dynamic list's object block, its own resource block, or within the handler for the query message. If the data is visual monikers, the data must appear within a resource block.

If the data is within a different block from the dynamic list, that block must be locked before its data can be accessed. Make sure to unlock the block after using the data.

Code Display 11-7 Creating the List of Data

```
/* Create the list of data (in this case within its own data block). This list of
 * data will be in form of text strings. */

@start ItemText, data;

@chunk char *listMonikers[NUM_COLORS] = {
    “Red”, “Yellow”, “Blue”, “Green”
};

@end ItemText

/* Create the query message handler. Each time the GenDynamicList needs to display
 * an item, it will call this querying message with the number (position) of the
 * item it needs to display. In this case, since the monikers are already in the
 * form of text strings, you can use MSG_GEN_DYNAMIC_LIST_REPLACE_ITEM_TEXT
 * to set up the moniker of the list item using the text string.
 * DEFINITION: void MSG_QUERY_COLOR_MONIKER(optr list, word item) */
```

```
@method MyProcessClass, MSG_QUERY_COLOR_MONIKER {
char *listString;
    /* The method will retrieve the entry’s text moniker from the array of
     * text monikers in listMonikers. It locks itemText, extracts the
     * text string, and uses the REPLACE_ITEM message to set the dynamic list
     * entry. */

    MemLock(HandleOf(@listMonikers));
    listString = listMonikers[item];

        /* Then send off the item string to the dynamic list
         * and unlock the data block. */
    @call listObject::MSG_GEN_DYNAMIC_LIST_REPLACE_ITEM_TEXT(item, listString);
    MemUnlock(HandleOf(@listMonikers));
}
```

■ MSG_GEN_DYNAMIC_LIST_REPLACE_ITEM_MONIKER

```
void    MSG_GEN_DYNAMIC_LIST_REPLACE_ITEM_MONIKER(@stack
        word                    item,
        word                    flags,
        word                    height,
        word                    width,
        word                    length,
        VisMonikerDataType      dataType,
        VisMonikerSourceType    sourceType,
        dword                   source);
```

This message sends a moniker to use for a particular list item and is usually sent in response to a moniker query by a dynamic list object. Within your query message handler (*GDLI_queryMsg*), you should access the data for the moniker to use and send off this message with the proper parameters. You may also use this message to change an item’s moniker, guaranteeing that it is updated if currently visible.

If you pass this message the flag RIMF_NOT_ENABLED, the moniker for the item will be disabled (usually grayed out).

This is an all-purpose message that allows many different source types and many ways to reference the data in these sources. If you are using lists of visual monikers or simple text strings and you do not to perform any fancy operations on this data, you may use MSG_GEN_DYNAMIC_LIST_REPLACE_ITEM_MONIKER_OPTR or MSG_GEN_DYNAMIC_LIST_REPLACE_ITEM_TEXT.

Source: The object handling a dynamic list's query message.

Destination: The GenDynamicList specified in the query message.

Parameters:
item The position of the item needing a moniker.

flags RIMF_NOT_ENABLED if the item's moniker should be shown disabled.

height If the item is a GString, the height in points.

width If the item is a GString, the width in points.

length Size of moniker data, in bytes. This value is ignored for VMST_OPTR. If the dataType is VMDT_TEXT and length is zero, the text moniker is assumed to be null-terminated. If the dataType is VMDT_GSTRING and the length is zero, the GString is assumed to end with a GR_END_GSTRING element.

dataType The **VisMonikerDataType** of the actual moniker.

sourceType The type of pointer referencing the moniker.

source A pointer to the moniker data.

Structures: The **VisMonikerDataType** and **VisMonikerSourcetype** are defined by **VisClass**. Their values are shown here for easy reference.

```
typedef ByteEnum VisMonikerDataType;
/*    VMDT_NULL
 *    VMDT_VIS_MONIKER
 *    VMDT_TEXT
 *    VMDT_GSTRING
 *    VMDT_TOKEN */
```

```
typedef ByteEnum VisMonikerSourceType;
/*    VMST_FPTR
 *    VMST_OPTR
 *    VMST_HPTR */
```

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_DYNAMIC_LIST_REPLACE_ITEM_MONIKER_OPTR

```
void        MSG_GEN_DYNAMIC_LIST_REPLACE_ITEM_MONIKER_OPTR(
            word   item,
            optr   moniker);
```

This message is a simplified version of MSG_GEN_DYNAMIC_LIST_REPLACE_ITEM_MONIKER. The moniker must be of type **VisMoniker** and referenced by an optr. No special operations (such as marking the item disabled) are possible with this message. In those cases, use MSG_GEN_DYNAMIC_LIST_REPLACE_ITEM_MONIKER.

Source: The object handling a dynamic list's query message.

Destination: The GenDynamicList object specified by the query message.

Parameters:

item	The position of the item to display.
optr	The optr of the **VisMoniker** to use.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_DYNAMIC_LIST_REPLACE_ITEM_TEXT

```
void        MSG_GEN_DYNAMIC_LIST_REPLACE_ITEM_TEXT(
            word   item,
            char   *text);
```

This message is a simplified version of MSG_GEN_DYNAMIC_LIST_REPLACE_ITEM_MONIKER. The moniker passed must be a null-terminated text string. No special operations (such as marking the item disabled) are possible with this message. In those cases, use MSG_GEN_DYNAMIC_LIST_REPLACE_ITEM_MONIKER

Source: The object handling a dynamic list's query message.

Destination: The GenDynamicList object specified by the query message.

Parameters:

item	The position of the item to display.
text	A pointer to the null-terminated text string.

Return: Nothing.

Interception: Generally not intercepted.

11.4.3 Altering Instance Data

You may alter the instance data of a GenDynamicList dynamically. Any of the appropriate GenItemGroup messages are valid for a GenDynamicList object except for MSG_GEN_ITEM_GROUP_GET_ITEM_OPTR.

11.4.3.1 Altering the Number of Items

```
MSG_GEN_DYNAMIC_LIST_INITIALIZE,
MSG_GEN_DYNAMIC_LIST_GET_NUM_ITEMS
```

In order to ensure the proper display of monikers in a dynamic list, the list object must know the total number of items that may be displayed. In most cases, you will set this number in the initial instance data for the object. In some cases, however, you will not initially know how many entries may be displayed; the total number of items will only be known after the data block is accessed.

In these cases, you should use MSG_GEN_DYNAMIC_LIST_INITIALIZE, MSG_GEN_DYNAMIC_LIST_ADD_ITEMS or MSG_GEN_DYNAMIC_LIST_REMOVE_ITEMS to dynamically set the *GDLI_numItems* instance field. You can retrieve the current number of items with MSG_GEN_DYNAMIC_LIST_GET_NUM_ITEMS.

■ MSG_GEN_DYNAMIC_LIST_INITIALIZE

```
void    MSG_GEN_DYNAMIC_LIST_INITIALIZE(
        word   numItems);
```

This message initializes a dynamic list, updating the display to reflect a new number of items or new data within the moniker list. If the list is already usable, this message invalidates all the current items and requests new monikers for all displayable items. The dynamic list will clear all previous selections; it will not scroll to any new selections. Use MSG_GEN_ITEM_GROUP_MAKE_ITEM_VISIBLE to scroll to a selected item.

This message is useful for cases when initially building a dynamic list without knowing the number of items; after you know the number of items, make sure to send the dynamic list this message.

Source: Unrestricted.

Destination: Any GenDynamicList object.

Parameters: *numItems* The new number of items for the dynamic list or GDLI_NO_CHANGE to keep the current number of selections.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_DYNAMIC_LIST_GET_NUM_ITEMS

```
word	MSG_GEN_DYNAMIC_LIST_GET_NUM_ITEMS();
```

This message returns the current number of items for a dynamic list. If no items are within the list, this message returns zero.

Source: Unrestricted.

Destination: Any GenDynamicList object.

Parameters: None.

Return: The number of items in the GenDynamicList (*GDLI_numItems*).

11.4.3.2 Adding and Removing Items Dynamically

MSG_GEN_DYNAMIC_LIST_ADD_ITEMS,
MSG_GEN_DYNAMIC_LIST_REMOVE_ITEMS,
MSG_GEN_DYNAMIC_LIST_REMOVE_ITEM_LIST

Part of the strength of a dynamic list is its ability to be easily updated with new data. MSG_GEN_DYNAMIC_LIST_ADD_ITEMS adds one or more new entries at any position in a dynamic list. The total number of items (*GDLI_numItems*) will be automatically incremented, and all entries after the added item will have their positions increased by one.

The constants GDLP_FIRST and GDLP_LAST indicate either the first or last item, respectively, and are available for use with these messages.

MSG_GEN_DYNAMIC_LIST_REMOVE_ITEMS removes one or more entries from any position within a dynamic list. The total number of items (*GDLI_numItems*) will be decremented, and all entries after the deleted item will have their positions decreased by one.

■ MSG_GEN_DYNAMIC_LIST_ADD_ITEMS

```
void      MSG_GEN_DYNAMIC_LIST_ADD_ITEMS(
          word   item,
          word   numItems);
```

This message adds an item after the position passed. All items that previously appeared at or after the position specified will have their positions incremented, and the total number of items will be incremented by *numItems*.

Usually, you will send this message after a change in the moniker data occurs. The moniker for the new item will be requested when it is made visible.

Source: Unrestricted.

Destination: Any GenDynamicList object.

Parameters: *item* The position of the first item to be added to the list. You may pass GDLP_FIRST to add the items at the beginning of the list or GDLP_LAST to add them at the end.

numItems The number of items to add.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_DYNAMIC_LIST_REMOVE_ITEMS

```
void      MSG_GEN_DYNAMIC_LIST_REMOVE_ITEMS(
          word   item,
          word   numItems);
```

This message removes the specified item(s) from a dynamic list at the position passed. All items that previously appeared after those deleted will have their positions decremented, and the total number of items will be decremented by the number of items removed.

Usually, you will send this message after a change in the moniker data. If the moniker for the deleted item is visible, a visual update will occur to reflect the list's new item positions. If the deleted item was selected, it will be deselected before being removed.

Source: Unrestricted.

Destination: Any GenDynamicList object.

Parameters: *itemPosition* The position of first item to be removed. GDLP_LAST or GDLP_FIRST specify the first or last position in the list.

numItems The number of items to be removed.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_DYNAMIC_LIST_REMOVE_ITEM_LIST

```
void    MSG_GEN_DYNAMIC_LIST_REMOVE_ITEM_LIST(
        word   *deletionList,
        word   numItems);
```

This message removes a list of items from a dynamic list. Therefore, unlike MSG_GEN_DYNAMIC_LIST_REMOVE_ITEMS, this message can remove separate items that occur at different positions in the list. The message updates other item positions appropriately, causing a visual refresh, if necessary, and updates the total number of items.

Source: Unrestricted.

Destination: Any GenDynamic List.

Parameters: *deletionList* Pointer to buffer storing positions (word values) of the items to be removed.

numItems Number of items being deleted.

Interception: Generally not intercepted.

11.5 GenBooleanGroups

GenBooleanGroupClass provides a list of items (GenBoolean objects) that may be individually set regardless of the state of other items in the list. The GenBooleanGroup is similar in behavior to a non-exclusive GenItemGroup in that each item may be selected independently of other selections. Unlike a non-exclusive GenItemGroup, however, GenBooleanGroup also allows your application to keep track of individual states for each item.

The GenBooleanGroup uses a word of data (16 bits) to keep track of the various states of its GenBoolean children. As a result, the maximum number of children allowed for a GenBooleanGroup object is 16. If you need non-exclusive behavior but have more than 16 children, you should either split the children between several GenBooleanGroups or use a non-exclusive GenItemGroup (or GenDynamicList).

GenBooleanClass is the basic item class for the **GenBooleanGroupClass**. It contains an entry that is set on or off. For example, a menu of font styles might have many entries such as “bold” or “italic,” each of which could be on or off, independent of each other. Each of those entries should be represented as a GenBoolean object.

11.5.1 GenBooleanGroup Instance Data

GenBooleanGroup and GenBoolean objects contain a variety of information. The bulk of the functionality of the list is contained in the GenBooleanGroup; the items themselves only contain identifying information that the list uses.

11.5.1.1 GenBooleanGroup Instance Data

The GenBooleanGroup contains instance fields relating to the state of its individual children. These instance fields are listed in Code Display 11-8.

Code Display 11-8 GenBooleanGroup Instance Fields

```
    @instance word      GBGI_selectedBooleans = 0;
    @instance word      GBGI_indeterminateBooleans = 0;
    @instance word      GBGI_modifiedBooleans = 0;
    @instance optr      GBGI_destination;
    @instance Message   GBGI_applyMsg = 0;

@vardata Message        ATTR_GEN_BOOLEAN_GROUP_STATUS_MSG;
@vardata optr           ATTR_GEN_BOOLEAN_GROUP_LINK;
    @reloc ATTR_GEN_BOOLEAN_GROUP_LINK, 0, optr;
@vardata void           ATTR_GEN_BOOLEAN_GROUP_INIT_FILE_BOOLEAN;

/* Hints */
```

```
@vardata void           HINT_BOOLEAN_GROUP_SCROLLABLE;
@vardata void           HINT_BOOLEAN_GROUP_MINIMIZE_SIZE;
@vardata void           HINT_BOOLEAN_GROUP_CHECKBOX_STYLE;
@vardata void           HINT_BOOLEAN_GROUP_TOOLBOX_STYLE;
```

GBGI_selectedBooleans is a record representing the selection state of the GenBooleanGroup's children. If a bit is set, the GenBoolean corresponding to that position is selected. Because of this field's size, no more than 16 GenBooleans may belong to a single GenBooleanGroup.

GBGI_indeterminateBooleans is a record representing the indeterminate state of the GenBooleanGroup's children. A GenBoolean is indeterminate if it cannot be considered either selected or unselected. If a bit is set, the GenBoolean corresponding to that position is indeterminate.

GBGI_modifiedBooleans is a record representing the modified state of the GenBooleanGroup's children. A GenBoolean is modified if its state (selected, deselected, or indeterminate) has changed since the last apply. If a bit is set, the GenBoolean corresponding to that position has been marked modified.

GBGI_destination contains the optr of the object or process that handles messages sent out by the GenBooleanGroup. This destination object will receive the apply action stored in *GBGI_applyMsg* whenever the user changes the GenBooleanGroup and that state change is later applied.

GBGI_applyMsg stores the apply action for the GenBooleanGroup to send out whenever a MSG_GEN_APPLY takes place, either through an immediate state change or through a delayed mode activation.

ATTR_GEN_BOOLEAN_GROUP_STATUS_MSG assigns a status message to your GenBooleanGroup. A status message allows objects to receive notice of a change in the BooleanGroup's current state without forcing an apply. This attribute is useful for cases where the state within your BooleanGroup may reflect information within other user interface objects and should therefore be updated whenever its user changes occur, regardless of the need to apply those changes.

ATTR_GEN_BOOLEAN_GROUP_LINK allows two or more GenBooleanGroups to be linked together and act as one BooleanGroup. Each BooleanGroup may contain one or more booleans with constant values unique to all other booleans within the linked BooleanGroups. The links should also be circular, with the last group pointing back to the first. All of these BooleanGroups should have identical states so they will work correctly. Use this attribute to circumvent undesirable geometry constraints. All of the GenBooleanGroups should have identical states.

ATTR_GEN_BOOLEAN_GROUP_INIT_FILE_BOOLEAN forces the GenBooleanGroup to use "true" and "false" values in the GEOS.INI file.

HINT_BOOLEAN_GROUP_SCROLLABLE implements the Boolean Group as a scrollable list, if that feature is supported by the specific UI. HINT_BOOLEAN_GROUP_MINIMIZE_SIZE instructs the UI to minimize the size of the Boolean Group. Depending on the specific UI, the Boolean group may be placed within a popup list or made scrollable.

HINT_BOOLEAN_GROUP_CHECKBOX_STYLE indicates that the Boolean Group should display its Booleans with the "checkbox" style, if that style is supported by the Specific UI. HINT_BOOLEAN_GROUP_TOOLBOX_STYLE indicates that the group should use the toolbox style, with a simple box around the Boolean's visual moniker. This style is the default for toolboxes in OSF/Motif, but not for other Booleans.

11.5.1.2 GenBoolean Instance Data

GenBooleanClass contains one additional instance field:

```
    @instance word    GBI_identifier;
```

GBI_identifier sets the identifying keyword (a 16-bit mask representing a specific bit—or bits—within a word-length bitfield) that uniquely identifies this GenBoolean. This identifier is used by the GenBooleanGroup parent to reference the Boolean. No two Booleans may specify common bits set, although a single Boolean's identifier may have several bits set.

11.5.2 GenBooleanGroup Usage

The GenBooleanGroup manages up to 16 GenBoolean objects. You must define a new (or use a pre-existing) constant to represent each GenBoolean within the GenBooleanGroup. Each constant represents a bit within a word-length record.

The GenBooleanGroup uses 16-bit records to designate its list of items; each bit in the record corresponds to an item. The GenBooleanGroup keeps three such records: *GBGI_selectedBooleans*, *GBGI_indeterminateBooleans*, and *GBGI_modifiedBooleans* to represent the selected state, the indeterminate state, and the modified state of each GenBoolean.

11.5.2.1 Selected Booleans

`GBI_identifier, GBGI_selectedBooleans`

GBI_identifier stores the identifying keyword for a GenBoolean item. The BooleanGroup object will always use this identifier when referring to GenBooleans. These identifiers are set up as constants representing bits in a 16-bit record. (See Code Display 11-9.)

Code Display 11-9 Setting Up a GenBooleanGroup

```
/* The identifiers for any particular GenBooleanGroup are
 * constant bits within a word-length bitfield. */

typedef WordFlags USCities;
#define USC_CHICAGO             0x0001          /* The first bit */
#define USC_NEW_YORK            0x0002          /* The second bit */
#define USC_SAN_FRANCISCO       0x0004          /* The third bit */

@object GenBooleanGroupClass CityList = {
    GI_comp = @Chicago, @NewYork, @SanFrancisco;
        /* The first and second bit in the selectedBooleans record are set to
         * indicate that Chicago and New York are initially selected. */
    GBGI_selectedBooleans = USC_CHICAGO | USC_NEW_YORK;
}

@object GenBooleanClass Chicago = {
    GI_visMoniker = “Chicago”;
    GBI_identifier = USC_CHICAGO;
}

@object GenBooleanClass NewYork = {
    GI_visMoniker = “New York”;
    GBI_identifier = USC_NEW_YORK;
}
```

```
@object GenBooleanClass SanFrancisco = {
    GI_visMoniker = “San Francisco”;
    GBI_identifier = USC_SAN_FRANCISCO;
}
```

GBGI_selectedBooleans stores a 16-bit record corresponding to the selection state of a GenBooleanGroup’s children. Selecting or deselecting the Booleans will alter the bit representing that GenBoolean child in this instance field. By setting selections in this field, you can specify initial selections for the GenBooleanGroup to initially appear set.

11.5.2.2 Indeterminate and Modified Booleans

`GBGI_indeterminateBooleans, GBGI_modifiedBooleans`

GBGI_indeterminateBooleans stores a 16-bit record corresponding to the indeterminate state of a GenBooleanGroup’s children. Your BooleanGroup may appear initially with GenBooleans in an indeterminate state. As opposed to a GenItemGroup, where either the entire list is indeterminate or not, individual Booleans may be indeterminate on their own accord.

A GenBoolean may be indeterminate while it is either selected or unselected. Making an item indeterminate will not change its selected state, though the item’s visual representation might change. (In OSF/Motif, an indeterminate selection appears “off” whether or not it is also selected.)

GBGI_modifiedBooleans stores a 16-bit record corresponding to the modified state of a GenBooleanGroup’s children. Any GenBooleans that have been modified since the occurrence of the last apply action will be marked modified. In general, you should not need to set any GenBooleans initially modified.

Whenever the GenBooleanGroup receives notice to “apply” any changes, it will check whether any Booleans have been modified. If any have, the GenBooleanGroup will send its apply action to its destination.

In immediate mode, any change to the GenBoolean’s state automatically sets the corresponding bit in the BooleanGroup’s *GBGI_modifiedBooleans* instance data and also sends out MSG_GEN_APPLY. Therefore, in immediate mode, any user changes will cause the apply action to be sent out immediately. In delayed mode, the corresponding bits in the *GBGI_modifiedBooleans* are set, but MSG_GEN_APPLY will be sent out at a later time. The apply action will only occur when the user forces it (such as clicking on an "apply" trigger) or when the application forces it (with MSG_GEN_APPLY).

11.5.2.3 Destination

```
GBGI_applyMsg, GBGI_destination,
GEN_BOOLEAN_GROUP_APPLY_MSG,
ATTR_GEN_SEND_APPLY_MSG_ON_APPLY_EVEN_IF_NOT_MODIFIED
```

When the GenBooleanGroup receives notice to apply any changes made in the list, it will first determine whether the group has been modified (by checking *GBGI_modifiedBooleans*). If any GenBoolean has been modified, the BooleanGroup will send the apply message stored in *GBGI_applyMsg* to the destination stored in *GBGI_destination*.

GBGI_applyMsg should contain the message you wish the GenBooleanGroup to send out whenever changes should be applied. This message should be based on the prototype GEN_BOOLEAN_GROUP_APPLY_MSG so the proper arguments are passed to its handler. *GBGI_destination* should contain the object or process that will handle the message sent out by this group.

You can force an apply to occur by marking GenBooleans modified with MSG_GEN_BOOLEAN_GROUP_SET_MODIFIED_STATE and sending MSG_GEN_APPLY to the GenBooleanGroup. You can also override the default behavior, forcing the object to send out its apply action on all applies, with ATTR_GEN_SEND_APPLY_MSG_ON_APPLY_EVEN_IF_NOT_MODIFIED in the object's instance data.

■ GEN_BOOLEAN_GROUP_APPLY_MSG

```
void     GEN_BOOLEAN_GROUP_APPLY_MSG(
         word   selectedBooleans,
         word   indeterminateBooleans,
         word   modifiedBooleans);
```

Use this prototype to define the apply message for your GenBooleanGroup (stored in *GBGI_applyMsg*). This prototype ensures that the message passes the correct parameters to your message handler.

Source: Your GenBooleanGroup object.

Destination: The destination of your GenBooleanGroup.

Parameters: *selectedBooleans* The currently selected GenBooleans (the contents of the *GBGI_selectedBooleans* instance field).

indeterminateBooleans
The current indeterminate GenBooleans (the contents of the *GBDI_indeterminateBooleans* instance field).

modifiedBooleans The current modified GenBooleans (the contents of the *GBGI_modifiedBooleans* instance field).

Return: Nothing.

11.5.2.4 Sending the Status Messages

GEN_BOOLEAN_GROUP_STATUS_MSG,
ATTR_GEN_BOOLEAN_GROUP_STATUS_MSG,
MSG_GEN_BOOLEAN_GROUP_SEND_STATUS_MSG

In delayed mode, GenBooleanGroups will only send out their apply action if an apply occurs. If you wish your application to receive notice whenever any user changes take place in the list's state, even if those changes should not be "applied," you may set up a status message in the object's instance data.

Set a message based on the prototype GEN_BOOLEAN_GROUP_STATUS_MSG. Set ATTR_GEN_BOOLEAN_GROUP_STATUS_MSG to this message in your object declaration, and provide a handler for this message in the class.

■ GEN_BOOLEAN_GROUP_STATUS_MSG

```
void      GEN_BOOLEAN_GROUP_STATUS_MSG(
          word   selectedBooleans,
          word   indeterminateBooleans,
          word   changedBooleans);
```

Use this prototype to define a status message for your GenBooleanGroup (stored in the ATTR_GEN_BOOLEAN_GROUP_STATUS_MSG vardata). This prototype ensures that the message passes the correct parameters to your message handler.

Source: Your GenBooleanGroup object.

Destination: The destination of your GenBooleanGroup object.

Parameters: *selectedBooleans* The currently selected GenBooleans (the contents of the *GBGI_selectedBooleans* instance field).

indeterminateBooleans The current indeterminate GenBooleans (the contents of the *GBDI_indeterminateBooleans* instance field).

changedBooleans The GenBooleans that caused the status message to be sent out (not necessarily the contents of the *GBGI_modifiedBooleans* instance field).

Return: Nothing.

■ MSG_GEN_BOOLEAN_GROUP_SEND_STATUS_MSG

```
void      MSG_GEN_BOOLEAN_GROUP_SEND_STATUS_MSG(
          word   changedBooleans);
```

This message causes the GenBooleanGroup to send out its status message stored in ATTR_GEN_ITEM_GROUP_STATUS_MSG. The message should include the current state of the GenBooleanGroup. (Base the status message on the prototype GEN_BOOLEAN_GROUP_STATUS_MSG to ensure this.) To mark individual GenBooleans as changed since the last apply action, pass this message a record of the changed GenBooleans.

Source: Unrestricted.

Destination: Any GenBooleanGroup object with a status message.

Parameters: *changedBooleans* A record representing the GenBooleans that have changed to cause this GenBooleanGroup to send out its status message.

Return: Nothing.

Interception: Generally not intercepted.

11.5.2.5 BooleanGroup Links

ATTR_GEN_BOOLEAN_GROUP_LINK

Just as you may create links between GenItemGroups, you may also create links between BooleanGroups. For more complete information on the purpose and effects of linked groups, see “GenItemGroup Links” on page 752.

To enable linked BooleanGroups, you may set up small sub-sets of GenBooleans as separate GenBooleanGroups. Within each BooleanGroup, include a link to the next BooleanGroup with ATTR_GEN_BOOLEAN_GROUP_LINK. Links should be circular; i.e., if three BooleanGroups should be linked, the first group should point to the second, the second to the third, and the third to the first. The three linked BooleanGroups will act as one entity.

11.5.2.6 Scrolling Boolean Groups

```
HINT_BOOLEAN_GROUP_SCROLLABLE,
MSG_GEN_BOOLEAN_GROUP_MAKE_BOOLEAN_VISIBLE
```

Boolean group lists, like item group lists, can be scrollable. To make a Boolean list scrollable, set HINT_BOOLEAN_GROUP_SCROLLABLE. To make a particular item in a scrollable Boolean list visible, use the message MSG_GEN_BOOLEAN_GROUP_MAKE_BOOLEAN_VISIBLE.

■ MSG_GEN_BOOLEAN_GROUP_MAKE_BOOLEAN_VISIBLE

```
void    MSG_GEN_BOOLEAN_GROUP_MAKE_BOOLEAN_VISIBLE(
        word   identifier);
```

This message, when sent to a scrollable BooleanGroup object, will make the specified GenBoolean visible in the list. If the passed identifier is unknown or is not usable, the BooleanGroup will do nothing.

Source: Unrestricted.

Destination: Any scrollable, usable GenBooleanGroup.

Parameters: *identifier* The item number of the GenBoolean to be made visible.

Return: Nothing.

Interception: Generally not intercepted.

11.5.3 Altering Instance Data

Both the BooleanGroups and the Booleans themselves may be dynamically changed to alter the behavior of the list objects.

11.5.3.1 Altering the Identifiers

MSG_GEN_BOOLEAN_GET_IDENTIFIER,
MSG_GEN_BOOLEAN_SET_IDENTIFIER

Each GenBoolean object contains an instance field to store the identifier of that object. It is this identifier that the parent GenBooleanGroup object uses when referring to individual GenBoolean items. You may change these identifiers dynamically if you wish.

To get a Boolean's identifier, send it MSG_GEN_BOOLEAN_GET_IDENTIFIER. To change a GenBoolean's identifier, send the object a MSG_GEN_BOOLEAN_SET_IDENTIFIER, passing the new identifier. You can send the GenBoolean object this message at any time, though sending it while the item is still GS_USABLE would be unusual.

■ MSG_GEN_BOOLEAN_GET_IDENTIFIER

```
word      MSG_GEN_BOOLEAN_GET_IDENTIFIER();
```

This message returns the identifier for the GenBoolean item.

Source: Unrestricted.

Destination: Any GenBoolean object.

Parameters: None.

Return: Identifier stored in the object's *GBI_identifier* instance field.

Interception: Generally not intercepted.

■ MSG_GEN_BOOLEAN_SET_IDENTIFIER

```
void      MSG_GEN_BOOLEAN_SET_IDENTIFIER(
          word   identifier);
```

This message sets a new identifier for a GenBoolean object. No change is made to the GenBooleanGroup object. You must make sure that you are not referencing the GenBoolean with its old identifier before sending this message.

Source: Unrestricted.

Destination: Any GenBoolean object.

Parameters: *identifier* A word representing the constant of the new identifier. This should be a mask representing a bit within a 16-bit record.

Return: Nothing.

Interception: Generally not intercepted.

Warnings: The GenBooleanGroup must not be referring to the object when the GenBoolean receives this message. You should also not refer to the GenBoolean by its old identifier from then on.

11.5.3.2 Altering the Group State

You may alter the state of a GenBooleanGroup's children in two ways: First, you can set the entire group's state by setting the 16-bit records in the GenBooleanGroup's instance data. Second, you may set individual bits representing an item's state individually.

Setting the State Collectively

```
MSG_GEN_BOOLEAN_GROUP_GET_SELECTED_BOOLEANS,
MSG_GEN_BOOLEAN_GROUP_SET_GROUP_STATE,
MSG_GEN_BOOLEAN_GROUP_GET_INDETERMINATE_BOOLEANS,
MSG_GEN_BOOLEAN_GROUP_SET_GROUP_INDETERMINATE_STATE,
MSG_GEN_BOOLEAN_GROUP_GET_MODIFIED_BOOLEANS,
MSG_GEN_BOOLEAN_GROUP_SET_GROUP_MODIFIED_STATE
```

To retrieve the contents of the *GBGI_selectedBooleans* instance field, send the BooleanGroup a MSG_GEN_BOOLEAN_GROUP_GET_SELECTED_BOOLEANS. This message returns a 16-bit record which you may then examine and alter. You can set the selection state of a BooleanGroup's children by sending the group a MSG_GEN_BOOLEAN_GROUP_SET_GROUP_STATE.

To retrieve the contents of the *GBGI_indeterminateBooleans* instance field, use MSG_GEN_BOOLEAN_GROUP_GET_INDETERMINATE_BOOLEANS. This message returns a 16-bit record. You can set the indeterminate state with MSG_GEN_BOOLEAN_GROUP_SET_GROUP_INDETERMINATE_STATE.

To retrieve the contents of the *GBGI_modifiedBooleans* instance field, use MSG_GEN_BOOLEAN_GROUP_GET_MODIFIED_BOOLEANS. This message returns a16-bit record which you may then examine and alter. You can set the selection state of a BooleanGroup's children by sending the group a MSG_GEN_BOOLEAN_GROUP_SET_GROUP_MODIFIED_STATE.

■ MSG_GEN_BOOLEAN_GROUP_GET_SELECTED_BOOLEANS

```
word        MSG_GEN_BOOLEAN_GROUP_GET_SELECTED_BOOLEANS();
```

This message returns the current state of all GenBooleans in a GenBooleanGroup (selected or unselected) by returning the contents of the *GBGI_selectedBooleans* instance field.

Source: Unrestricted.

Destination: Any GenBooleanGroup object.

Parameters: None.

Return: A record indicating the GenBooleans selected.

Interception: Generally not intercepted.

■ MSG_GEN_BOOLEAN_GROUP_GET_INDETERMINATE_BOOLEANS

```
word        MSG_GEN_BOOLEAN_GROUP_GET_INDETERMINATE_BOOLEANS();
```

This message returns the indeterminate items of a GenBooleanGroup by returning the contents of the *GBDI_indeterminateBooleans* instance field.

Source: Unrestricted.

Destination: Any GenBooleanGroup object.

Parameters: None.

Return: A record indicating the current indeterminate booleans.

Interception: Generally not intercepted.

■ MSG_GEN_BOOLEAN_GROUP_SET_GROUP_STATE

```
void        MSG_GEN_BOOLEAN_GROUP_SET_GROUP_STATE(
            word   selectedBooleans,
            word   indeterminateBooleans);
```

This message sets new selections and indeterminate states for a GenBooleanGroup (altering the contents of the *GBGI_selectedBooleans* and *GBGI_indeterminateBooleans* instance field). You should pass the records indicating the GenBooleans to select and/or mark indeterminate. Any GenBooleans which are not set will be cleared.

Source: Unrestricted.

Destination: Any GenBooleanGroup object.

Parameters: *selectedBooleans*   The GenBooleans which should be selected.

indeterminateBooleans
The GenBooleans which should be marked indeterminate.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_BOOLEAN_GROUP_GET_MODIFIED_BOOLEANS

11.5

```
word        MSG_GEN_BOOLEAN_GROUP_GET_MODIFIED_BOOLEANS();
```

This message returns selections marked modified within a GenBooleanGroup since the last apply (by returning the contents of the *GBGI_modifiedBooleans* instance field).

Source: Unrestricted.

Destination: Any GenBooleanGroup object.

Parameters: None.

Return: The current modified Booleans.

Interception: Generally not intercepted.

■ MSG_GEN_BOOLEAN_GROUP_SET_GROUP_MODIFIED_STATE

```
void        MSG_GEN_BOOLEAN_GROUP_SET_GROUP_MODIFIED_STATE(
            word   setBooleans,
            word   clearBooleans);
```

GenBooleans are normally marked modified any time their state is altered, and they are marked not modified after these changes have been applied. You may also mark GenBooleans modified (or mark them nor modified) with this message. You should pass this message both the GenBooleans to set and the GenBooleans to clear (in the *GBGI_modifiedBooleans* instance field).

Source: Unrestricted.

Destination: Any GenBooleanGroup object.

Parameters: *setBooleans* A record indicating the GenBooleans to be set modified in *GBGI_modifiedBooleans*.

clearBooleans A record (mask) indicating the GenBooleans to be set not modified (cleared) in *GBGI_modifiedBooleans*.

Return: Nothing.

◆Objects

Interception: Generally not intercepted.

Setting the Booleans Individually

```
MSG_GEN_BOOLEAN_GROUP_GET_BOOLEAN_OPTR,
MSG_GEN_BOOLEAN_GROUP_IS_BOOLEAN_SELECTED,
MSG_GEN_BOOLEAN_GROUP_SET_BOOLEAN_STATE,
MSG_GEN_BOOLEAN_GROUP_IS_BOOLEAN_INDETERMINATE,
MSG_GEN_BOOLEAN_GROUP_SET_BOOLEAN_INDETERMINATE_STATE,
MSG_GEN_BOOLEAN_GROUP_IS_BOOLEAN_MODIFIED,
MSG_GEN_BOOLEAN_GROUP_SET_BOOLEAN_MODIFIED_STATE
```

You can retrieve the optr of a particular GenBoolean object by sending the BooleanGroup MSG_GEN_BOOLEAN_GROUP_GET_ITEM_OPTR, passing it the identifier of the boolean in question. You can then use that optr to directly communicate with the object for purposes such as enabling/disabling, setting not usable, etc.

To check on whether a GenBoolean is currently selected, send the BooleanGroup MSG_GEN_BOOLEAN_GROUP_IS_BOOLEAN_SELECTED, passing it the identifier to check. To set the selection state of a Boolean, send the BooleanGroup MSG_GEN_BOOLEAN_GROUP_SET_BOOLEAN_STATE, passing it the identifier and the selection state (on or off).

To check on the indeterminate state of a GenBoolean, send the BooleanGroup MSG_GEN_BOOLEAN_GROUP_IS_BOOLEAN_INDETERMINATE, passing the identifier to check. To set the indeterminate state of a Boolean, send MSG_GEN_BOOLEAN_GROUP_SET_BOOLEAN_INDETERMINATE_STATE, passing it the identifier and the indeterminate state to set the Boolean to.

To check on the modified state of a GenBoolean, send the BooleanGroup MSG_GEN_BOOLEAN_GROUP_IS_BOOLEAN_MODIFIED, passing the identifier to check. To set the modified state of a Boolean, send MSG_GEN_BOOLEAN_GROUP_SET_BOOLEAN_MODIFIED_STATE, passing it the identifier and the modified state.

■ MSG_GEN_BOOLEAN_GROUP_GET_BOOLEAN_OPTR

```
optr      MSG_GEN_BOOLEAN_GROUP_GET_BOOLEAN_OPTR(
          word   identifier);
```

This message returns the optr of the requested GenBoolean. You may then use this optr to alter that Boolean's state individually. If the GenBoolean with the matching identifier is not found, a null optr will be returned.

Source: Unrestricted.

Destination: Any GenBooleanGroup object.

Parameters: *identifier* The identifier of the GenBoolean to look for.

Return: The optr of the requested GenBoolean object (or a null optr if none is found).

Interception: Generally not intercepted.

■ MSG_GEN_BOOLEAN_GROUP_IS_BOOLEAN_SELECTED

```
Boolean   MSG_GEN_BOOLEAN_GROUP_IS_BOOLEAN_SELECTED(
          word   identifier);
```

This message checks whether the GenBoolean with the passed identifier is selected.

Source: Unrestricted.

Destination: Any GenBooleanGroup object

Parameters: *identifier* The identifier of the GenBoolean to check.

Return: Will return *true* if GenBoolean is selected, *false* if GenBoolean is unselected.

Interception: Generally not intercepted.

■ MSG_GEN_BOOLEAN_GROUP_SET_BOOLEAN_STATE

```
void      MSG_GEN_BOOLEAN_GROUP_SET_BOOLEAN_STATE(
          word               identifier,
          Boolean            state);
```

This message sets an individual GenBoolean's selection state, leaving other GenBooleans unaffected. You may alter the selection state of individual GenBooleans even if they are not usable, not enabled, or not even within the BooleanGroup. (This is useful for linked BooleanGroups.)

Source: Unrestricted.

Destination: Any GenBooleanGroup.

Parameters: *identifier* The identifier of the GenBoolean.

state TRUE to mark selected, FALSE to mark unselected.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_BOOLEAN_GROUP_IS_BOOLEAN_INDETERMINATE

```
Boolean   MSG_GEN_BOOLEAN_GROUP_IS_BOOLEAN_INDETERMINATE(
          word   identifier);
```

This message checks whether the GenBoolean with the passed identifier is indeterminate (by examining the *GBDI_indeterminateBooleans* instance field).

Source: Unrestricted.

Destination: Any GenBooleanGroup object.

Parameters: *identifier* The identifier of the GenBoolean to check.

Return: Will return *true* if the GenBoolean is indeterminate, *false* if is not.

Interception: Generally not intercepted.

■ MSG_GEN_BOOLEAN_GROUP_SET_BOOLEAN_INDETERMINATE_STATE

```
void      MSG_GEN_BOOLEAN_GROUP_SET_BOOLEAN_INDETERMINATE_STATE(
          word                  identifier,
          Boolean               indeterminateState);
```

This message sets the indeterminate state of an individual GenBoolean without affecting any other GenBooleans. You may mark GenBooleans indeterminate even if they are not usable, not enabled, or not even present within the BooleanGroup.

Source: Unrestricted.

Destination: Any GenBooleanGroup object.

Parameters: *identifier* The identifier of the GenBoolean.

indeterminateState

Pass TRUE to mark the GenBoolean indeterminate, FALSE to mark it not indeterminate.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_BOOLEAN_GROUP_IS_BOOLEAN_MODIFIED

```
Boolean  MSG_GEN_BOOLEAN_GROUP_IS_BOOLEAN_MODIFIED(
         word   identifier);
```

This message checks whether the GenBoolean with the passed identifier has been modified (by examining the *GBGI_modifiedBooleans* instance field).

Source: Unrestricted.

Destination: Any GenBooleanGroup object.

11.5

Parameters: *identifier* The identifier of the GenBoolean to check.

Return: Will return *true* if boolean has been modified, *false* if it has not.

Interception: Generally not intercepted.

■ MSG_GEN_BOOLEAN_GROUP_SET_BOOLEAN_MODIFIED_STATE

```
void     MSG_GEN_BOOLEAN_GROUP_SET_BOOLEAN_MODIFIED_STATE(
         word                identifier,
         Boolean             modifiedState);
```

This message sets the modified state of an individual GenBoolean without affecting any other GenBooleans. You may mark GenBooleans modified even if they are not usable, not enabled, or not even present within the BooleanGroup.

Source: Unrestricted.

Destination: Any GenBooleanGroup object.

Parameters: *identifier* The identifier of the GenBoolean.

modifiedState TRUE to mark the GenBoolean modified, FALSE to mark it not modified.

Return: Nothing.

Interception: Generally not intercepted.

11.5.3.3 Altering the Destination

```
MSG_GEN_BOOLEAN_GROUP_GET_DESTINATION,
MSG_GEN_BOOLEAN_GROUP_SET_DESTINATION,
```

◆Objects

MSG_GEN_BOOLEAN_GROUP_GET_APPLY_MSG,
MSG_GEN_BOOLEAN_GROUP_SET_APPLY_MSG

To retrieve the GenBooleanGroup's destination stored in its *GBGI_destination* instance field or the message stored in its *GBGI_applyMsg* instance field, use MSG_GEN_BOOLEAN_GROUP_GET_DESTINATION or MSG_GEN_BOOLEAN_GROUP_GET_APPLY_MSG.

To set the GenBooleanGroup's destination to an object or process, send the list a MSG_GEN_BOOLEAN_GROUP_SET_DESTINATION, passing it the optr of the new destination. This object will thereafter handle all apply actions sent by the GenBooleanGroup.

To set the GenBooleanGroup's apply action to a new message, send the list a MSG_GEN_BOOLEAN_GROUP_SET_APPLY_MSG, passing it the message to use.

■ MSG_GEN_BOOLEAN_GROUP_GET_DESTINATION

```
optr    MSG_GEN_BOOLEAN_GROUP_GET_DESTINATION();
```

This message returns the current destination object for the BooleanGroup, as specified in the *GBGI_destination* instance field.

Source: Unrestricted.

Destination: Any GenBooleanGroup object.

Parameters: None.

Return: The optr of the current destination object.

Interception: Generally not intercepted.

■ MSG_GEN_BOOLEAN_GROUP_SET_DESTINATION

```
void    MSG_GEN_BOOLEAN_GROUP_SET_DESTINATION(
        optr   dest);
```

This message sets a new destination object for a BooleanGroup. The apply and status messages of this BooleanGroup will be sent to this new destination.

Source: Unrestricted.

Destination: Any GenBooleanGroup.

Parameters: *dest* The optr of the new destination object; this will be put into *GBGI_destination*.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_BOOLEAN_GROUP_GET_APPLY_MSG

```
Message   MSG_GEN_BOOLEAN_GROUP_GET_APPLY_MSG();
```

This message returns the current apply action in the GenBooleanGroup's *GBGI_applyMsg* instance field.

11.5

Source: Unrestricted.

Destination: Any GenBooleanGroup.

Parameters: None.

Return: The current apply message.

Interception: Generally not intercepted.

■ MSG_GEN_BOOLEAN_GROUP_SET_APPLY_MSG

```
void      MSG_GEN_BOOLEAN_GROUP_SET_APPLY_MSG(
          Message             message);
```

This message sets a new apply message in the GenBooleanGroup's *GBGI_applyMsg* instance field.

Source: Unrestricted.

Destination: Any GenBooleanGroup object.

Parameters: *message* The new apply message.

Return: Nothing.

Return: Generally not intercepted.

Generic UI Controllers

12

Many applications will create their menus and dialog boxes from scratch, using various GenInteraction and GenTrigger objects. Several system components, however, use a standard base format across all applications. For these components, GEOS provides a number of controller objects.

A controller object is one that provides UI gadgetry to control a certain set of features. For example, the GenViewControl object provides all the UI—including menus, dialogs, and tool bars, if appropriate—to alter a GenView's scaling and other features. Many libraries such as the text library provide controller objects that make the use of the library almost trivial.

To make inclusion of these controllers even easier, the UI offers a GenToolControl; this object provides all the functionality and UI gadgetry to allow the user to configure his own toolbars and menus from the tools provided by a controller.

Finally, developers writing libraries or suites of applications may wish to create their own controller classes. This is neither difficult nor complex, though it will be done primarily by programmers writing libraries for use by many applications.

This chapter describes what controllers are and how they work. It also discusses which controllers are available in the system, as well as the GenToolControl object and how it's used. Finally, it details how to create your own controller classes. In order to fully understand this chapter, you should be familiar first with the generic UI, menus and dialog boxes, and the General Change Notification mechanism.

12.1 Controller Features and Functions

Controller objects let you include a library's features in your applications with limited work. The controller provides the basic UI gadgetry and feature management functions so you don't have to.

Every library has a set of features that will be used by all or most of the library's users. For example, applications which use the text objects (GenText and VisText) are likely to allow the user to alter justification, font, and text style. Rather than forcing every application to create its own menus and dialogs and lists for managing these features, the text library exports a number of controllers that can be included in the application's UI; these controllers provide and manage the menu items and lists for the user. They also interact directly with the text objects to change the attributes affected.

12.1

12.1.1 Controller Features

Controller objects provide the following features and benefits to application and library programmers:

- Ease of use
  Controllers are intended to be "plug-and-play" objects. Programmers simply include the controller to get all the appropriate UI gadgetry.

- Consistency of appearance
  All applications that use a particular controller will appear consistent to the user. The controller knows the appropriate structure of its menus and dialogs, and all such controllers will appear the same to the user, increasing the application's ease of use.

- Automatic tool management
  By using controllers along with a GenToolControl object, an application can allow the user to configure his own tool set. Controllers can have their UI placed along a display's edge, in the normal menus, or in a floating tool box; the GenToolControl allows users to decide where the tools will appear without affecting the application at all.

- Extendability of features
  Any application that uses a controller can automatically include future features of a library. For example, if the GenView object were upgraded in the future to allow new features, the GenViewControl object would likely also be updated. Any application including the GenViewControl would automatically get the new UI and features without recompilation. Applications that do not use controllers will not automatically gain the benefits of upgraded libraries.

The GenControl is most useful for simple controllers. It is based on core UI objects (GenTrigger, GenInteraction, etc.) and is subclassed from **GenInteractionClass**. Applications that will have extremely complex controller-type functions may want to write their controllers from scratch—this is discussed in “Creating Your Own Controllers” on page 810.

12.1.2 How Controllers Work

When you use a controller object, you are really using three different types of objects: The controller itself, the UI gadgetry it creates and manages, and the controlled object (sometimes called a “data item”) it acts on. For the most part, you will not have to understand these relationships unless you are creating your own controllers with **GenControlClass**.

To explain how a controller works with both of these other objects, this section uses a simple application that uses a single controller. The controller used is a PointSizeControl object, exported by the text library and used with the GenText and VisText objects; in this example, it interacts with two GenText objects. The entire source code of the application’s **psctext.goc** file is given in Code Display 12-1 on page ◆ 792. This code display does not show the **psctext.gp** file because it is similar to Hello World’s parameters file.

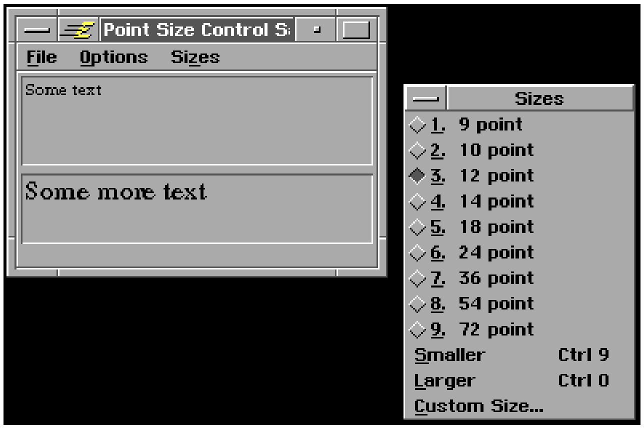


Figure 12-1 *Sample Controller Application*
The Sizes menu is managed by the PointSizeControl object.

A screen dump of the application is shown in Figure 12-1, and the entire object tree declared in the application is shown in Figure 12-2. The controller object (PSCTSizeControl) declares and sets up the entire menu structure of the Sizes menu, so the menu items do not need to be declared in the application's **.goc** file.

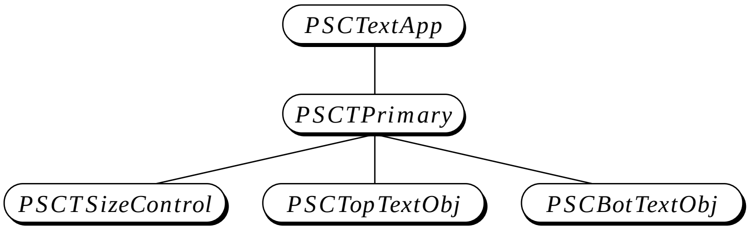


Figure 12-2 *The psctext.goc Object Tree*
This application has five generic objects, two of which are text objects acting in concert with the Point Size Control.

The interaction between the controller, the text objects, and the controller's UI objects is simple. When the controller is first loaded in from the application's Interface resource block, it also loads in the associated UI gadgetry for the Sizes menu. The menu items are added as children of the controller without the programmer having to do anything extra.

The data item objects (in this example the GenText objects), when loaded or created, create and set up special General Change Notification (GCN) lists. In the PSCText sample application, the text library sets up a GCN list for the notification type GAGCNLT_APP_TARGET_NOTIFY_CHAR_ATTR_CHANGE. When changes to the text object occur, the text object will send a message to this GCN list.

The controller object knows inherently which GCN lists it should be added to. When it is loaded, it adds itself to these lists. In the sample application, the PSCTSizeControl object will add itself automatically to the GCN list for GAGCNLT_APP_TARGET_NOTIFY_CHAR_ATTR_CHANGE.

The interactions between the three components are summarized in Figure 12-3. The application programmer needs to know very little about the interactions of these components; in fact, unless you are creating your own controller objects, you can get away with knowing only a very few things.

Building your own controller is detailed in “Creating Your Own Controllers” on page 810, and the basics of controller use, along with the source code for **psctext.goc**, are shown in “Using Controllers” on page 791.

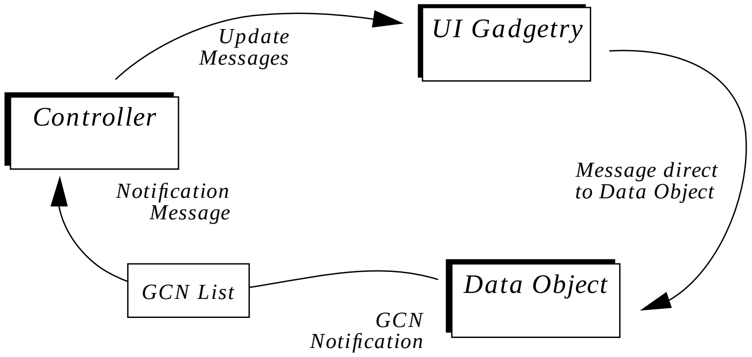


Figure 12-3 *Controller Interactions*

When the user clicks an item in the UI gadgetry, that gadget sends a message (e.g. MSG_VIS_TEXT_SET_CHAR_ATTR) to the Data Object. When the Data Object changes, it notifies the appropriate GCN list, which notifies the controller. The controller then enables/disables the UI gadgetry.

12.1.3 Using Controllers

As stated above, for an application to use a controller (or several controllers), the application programmer needs to know very little about the controller itself. Code Display 12-1 shows the *entire* code for the PSCText sample application; note that the programmer has to add no code (just object declarations) to create two editable text objects which can have selectable and settable point sizes.

The application programmer must do two essential things to use a controller object: First, you must include and set up the controller properly. In the example application, the PSCTSizeControl object is a menu; it could easily have been set up as GIV_DIALOG (typically, though, this particular controller is implemented as a menu). Because **GenControlClass** is a subclass of **GenInteractionClass**, you can set up the controller exactly like any other GenInteraction.

Second, you must set up your data objects to interact with the controller. The data objects should be written for this automatically; for example, the GenText objects automatically send the proper notification to the GCN mechanism to ensure that all appropriate controllers are notified of all changes. Note, however, that the GenText objects have to be set targetable before the controller can operate on them. Some data objects will be set up to work with their controllers automatically; others may need to have certain attributes set. The GenText objects, for example, are not by default targetable; the controller, however, sends its messages to the application's target object. If the GenTexts are not targetable, they will never gain the application's target and therefore will never react to the controller's requests.

To use a particular controller, you should learn about how that controller's data object needs to be set up. To help you with this, "Standard Controllers" on page 794 lists the various controllers included in the system and where they are documented.

Code Display 12-1 A Sample Controller Application (psctext.goc)

```
@include <stdapp.goh>
@include <ui.goh>
@include <Objects/Text/tCtrlC.goh>

/* The PSCText application's process class runs the application's primary
 * thread. For a description of GenProcessClass, see Hello World. */

@class PSCTextProcessClass, GenProcessClass;
@endc
@classdecl PSCTextProcessClass;

@start AppResource;
    /* The PSCTextApp object defines the application object for the
     * application. For full information, see Hello World.
     * The controller is also placed on the application's self-load-options
     * GCN list to ensure that it loads its options properly when returning
     * from saved state. This is shown below in the second of the two
     * GCN list declarations. */
@object GenApplicationClass PSCTextApp = {
    GI_visMoniker = "Point Size Control Sample Application";
    GI_comp = @PSCTPrimary;
    gcnList(MANUFACTURER_ID_GEOWORKS,GAGCNLT_WINDOWS) = @2PSCTPrimary;
    gcnList(MANUFACTURER_ID_GEOWORKS,GAGCNLT_SELF_LOAD_OPTIONS) = @PSCTSizeControl;
}
```

Objects

```
@end AppResource

@start Interface;

/* The PSCTPrimary object serves as the primary window of this sample application.
 * Its children are the Point Size Control object and the two GenText objects. */
@object GenPrimaryClass PSCTPrimary = {
    GI_comp = @PSCTSizeControl, @PSCTopTextObj, @PSCBotTextObj;
    HINT_SIZE_WINDOW_AS_DESIRED;
    HINT_ORIENT_CHILDREN_VERTICALLY;
}
```

```
/* The PSCTSizeControl object is the controller that provides all the point size
 * functionality. It will automatically create a menu called “Sizes” (due to the
 * GI_visMoniker) and all the entries of that menu.
 * This controller will work on whichever of the two GenText objects
 * (PSCGTopTextObj and PSCBotTextObj) is set the target; the controller’s UI
 * objects (the Sizes menu) send their messages directly to the target via the
 * TravelOption TO_TARGET. If a point size is selected as the user’s first action,
 * it will work on PSCTopTextObj because that is set up as the default target. */
@object PointSizeControlClass PSCTSizeControl = {
    GI_visMoniker = ’z’, “Sizes”;        /* Give the controller a name */
    GII_visibility = GIV_POPUP;          /* Make the controller a menu */
}

/* These two GenText objects are simple; they use only the defaults plus the
 * items shown here. Both must be set targetable (GA_TARGETABLE) to be included
 * in the target hierarchy; this is necessary when using controllers because of
 * the note in the above comment. The PSCTopTextObj is made the default focus
 * (to get keyboard input) and the default target (for controller operation). */
@object GenTextClass PSCTopTextObj = {
    GI_attrs = @default | GA_TARGETABLE;
        /* Initially, this text object uses the VisTextDefaultSize VTDS_12
         * (12 pts) and the VisTextDefaultFont VTDF_URW_ROMAN. You can use
         * the PointSizeControl object to change this point size. */
    ATTR_GEN_TEXT_DEFAULT_CHAR_ATTR =    ((VTDS_12 << VTDCA_SIZE_OFFSET) |
                                          VTDF_URW_ROMAN);
        /* Set the font mapping to none to turn off the defaults. */
    HINT_DEFAULT_FOCUS;
    HINT_DEFAULT_TARGET;
}
```

```
@object GenTextClass PSCBotTextObj = {
    GI_attrs = @default | GA_TARGETABLE;
    ATTR_GEN_TEXT_DEFAULT_CHAR_ATTR =    ((VTDS_12 << VTDCA_SIZE_OFFSET) |
                                          VTDF_URW_ROMAN);
}
@end Interface
```

12.2 Standard Controllers

This section lists the various controller classes exported by system objects and libraries. Many are part of the GEOS UI; others are part of particular libraries. Some are system-wide (such as the ColorSelectorClass); others are object-specific (such as the InkControllerClass).

All of these controllers are set up to work with **GenToolControlClass** and **GenToolGroupClass**. When a GenToolControl object is also included in an application, users may set up menus and floating toolboxes however they like with the various tools from the controller's lists. For more information on the GenToolControl and how it is used, see “Using Tools” on page 801 and “GenToolControlClass” on page 840.

- **GenToolControlClass**
  This controller class provides the UI for the user to select where other controller tools appear. Controller tools must be specified by the individual controller classes; **GenToolControlClass** simply manages them according to the user's specifications. Individual tools may be placed in a floating tool box, in an application's tool bar, alongside the active display, alongside the primary display, or other places. **GenToolControlClass** is described in full in “Using Controllers” on page 796.

- **GenEditControlClass**
  This controller class provides the Edit menu with the Undo, Cut, Copy, Paste, Select All, and Delete triggers. This controller will cooperate fully with most system objects such as the text, graphic object, spreadsheet, and Ink objects. The GenEditControl sends out MSG_META_CUT, MSG_META_COPY, etc., to these objects. **GenEditControlClass** is described in “The Clipboard,” Chapter 7 of the Concepts Book.

- **GenViewControlClass**
  This controller class provides a View menu with the functions for changing a GenView's scale, paging, scrolling, aspect ratio, and other features. It is described in full in "GenView," Chapter 9.

- **GenDisplayControlClass**
  This controller class provides the UI for a multiple display interface. It works with the GenDisplayGroup and GenDisplay objects and is described in full in "GenDisplay / GenPrimary," Chapter 4.

- **GenDocumentControlClass**
  This controller class provides all the functions of the File menu including New, Open, Use Template, Close, Save, Save As, and Revert. It also interacts with the GenDocument and GenDocumentGroup objects to manage how the user interacts with documents in general. **GenDocumentControlClass** is described in full in "GenDocument," Chapter 13.

- **GenPageControlClass**
  This controller class provides the UI for a user to go to a previous, next or specified page. It is described in full in "GenPageControlClass" on page 853.

- **PrintControlClass**, **PageSizeControlClass**
  Both of these controller classes are exported by the spool object library. Both are described in full in "The Spool Library," Chapter 17.

- **ImportControlClass**, **ExportControlClass**
  These controller classes provide the UI and functions necessary for the user to select files for importing and exporting. They are exported by the Impex library are described in full in "Impex Library," Chapter 16.

- **InkControlClass**
  This controller class provides the tool interface for working with Ink-related tools. It is exported by the pen library and is described in full in "Pen Object Library," Chapter 21.

- **ColorSelectorClass**
  This controller class provides the UI for a user to set a color using either the palette index or a set of RGB values, to set a draw mask, and to set an area fill pattern. **ColorSelectorClass** is described in "ColorSelectorClass" on page 844.

- **StyleSheetControlClass**
  This controller class provides the UI and functions necessary for a user to

define and change an application-specific style type and apply style changes. It works primarily with the Text library and graphic object library.

- **FloatFormatClass**
  This controller class provides the UI and functions to allow the user to format numerical values into text (for example scientific notation, dates and times, etc.) This controller is described in “The Float Format Controller” on page 855.
- Various Text Controller Classes
  The text library exports a number of controllers that are specific to text operations. These classes are described in “The Text Objects,” Chapter 10.
- Various GrObj Controller Classes
  The graphic object library exports a number of controller classes that are specific to operations on and with the GrObj. These are all detailed in “Graphic Object Library,” Chapter 18.
- Various Spreadsheet Controller Classes
  Several controller class provide UI and functions for the user to interact with the floating point library and the spreadsheet object. They are exported by the spreadsheet library and are documented in “Spreadsheet Objects,” Chapter 20.
- Various Ruler Controller Classes
  The ruler library exports a number of controllers that manage the user’s interactions and control over the ruler objects. These controller classes include **RulerTypeControlClass**, **GuideCreateControlClass**, and **RulerGridControlClass**. They are described in full in “Ruler Object Library,” Chapter 19.

12.3 Using Controllers

There are essentially two ways to use controller objects: The first, and the simplest, is to simply include an object of an existing controller class and the data object for which it’s designed (e.g. using a PointSizeControl object with a GenText object). The second, and quite more complex, is to create either your own data objects or your own controllers.

This section describes the basic components of a controller object on the most general level—how to include a pre-existing controller along with its data object. For full information on how the **GenControlClass** works and how a data object interacts with it, see “Creating Your Own Controllers” on page 810. This section focuses on how you can use a controller object; nearly all of this will be academic if you use a GenToolControl object in your application.

12.3.1 Using a Basic GenControl Object

To use a typical controller, you have to set up its instance data appropriately and set up the generic tree properly. This section describes the basics of **GenControlClass** instance data; for full information on **GenControlClass**, see section 12.4 on page 810.

Every controller has a default set of features and a default set of tools. When you use a controller in your application, you can use the default configuration, the configuration appropriate to the application’s user level, or a specific configuration. Most applications will want to use the second option; the controller queries the GenApplication object for its *GAI_appFeatures* record and determines from that which of its features and tools should be active.

Controllers can be manifested in any way the specific UI determines appropriate; three main ways, however, are menus or submenus, floating tool boxes (dialogs), and groupings along a tool bar in a window. These three modes correspond to the three manifestations of a typical GenInteraction object: GIV_POPUP (menus), GIV_DIALOG (dialogs and tool boxes), and GIV_SUB_GROUP (groupings of other generic objects). If you use a GenToolControl, you can let the user decide how the controller is displayed; otherwise, you will have to set it manually (as in the samples previously).

Every controller also has two sets of UI objects: The first set represents the UI objects used for menu entries (when the controller is in GIV_POPUP mode). The second set represents the tools that appear in a tool box (GIV_DIALOG mode) or tool bar (GIV_SUB_GROUP mode). Tools are almost always functionally redundant to the “menu” feature set. Because the tool and the menu UI resources contain different objects, the UI objects can exist in any combination of interactable states—for example, a particular feature could be in the menu only, in the tool box only, in both, or in neither. The feature can not, however, be in both a tool bar and a tool box at the same time because the set of tools can be grouped in only one location.

12.3.1.1 Using Normal Features

ATTR_GEN_CONTROL_REQUIRE_UI, ATTR_GEN_CONTROL_PROHIBIT_UI

GenControlClass has no controller features of its own; instead, each controller class must define the features it supports in both the menu implementation and the tool implementation. Any controller object is free to determine which of the features it will support and which it will not.

To set individual features on or off for a controller object in your application, use the vardata fields ATTR_GEN_CONTROL_REQUIRE_UI and ATTR_GEN_CONTROL_PROHIBIT_UI; these specify which features will be on and which will be off. For listings of a controller class' features, you must see the description of the individual class.

The example in Code Display 12-2 extends the example from earlier in the chapter to turn on only the 10-, 12-, and 24-point as well as the "Larger" and "Smaller" features. It turns off all other features.

Code Display 12-2 Declaring a Controller's Features

```
/* This example is based on that of Code Display 12-1 on page ◆ 792. It shows what
 * would change in order to turn on only the 10-, 12-, and 24-point as well as the
 * "Larger" and "Smaller" features. Note that only the Controller object must
 * be altered. */

@object PointSizeControlClass PSCTSizeControl = {
    GI_visMoniker = 'z', "Sizes";        /* Give the controller a name */
    GII_visibility = GIV_POPUP;          /* Make the controller a menu */
        /* The following attribute defines which of the controller's
         * features are to be supported. These menu items will appear
         * in the controller's Size menu. */
    ATTR_GEN_CONTROL_REQUIRE_UI = (PSCF_10 | PSCF_12 | PSCF_24 |
                            PSCF_SMALLER | PSCF_LARGER);
        /* The following attribute defines which of the controller's
         * features will not be supported. These menu items will not
         * appear in the controller’s Size menu. */
    ATTR_GEN_CONTROL_PROHIBIT_UI = (PSCF_14 | PSCF_18 | PSCF_36 | PSCF_54 |
                                 PSCF_72 | PSCF_CUSTOM_SIZE);
}
```

12.3.1.2 Adding Application-Specific UI Gadgetry

ATTR_GEN_CONTROL_APP_UI, ATTR_GEN_CONTROL_APP_TOOLBOX_UI

Occasionally, an application will want to add its own UI gadgetry to a controller. This is not the same as changing the controller’s functionality—to do that, you would need to subclass the controller class. Rather, this entails simply specifying a group of generic UI objects (e.g. a GenInteraction and some GenTriggers) that will be included with the controller’s UI objects.

The **GenControlClass** vardata attribute ATTR_GEN_CONTROL_APP_UI allows you to specify a generic tree that will be added as a child of the controller. The top node of this tree must be an object that can be a child of a GenInteraction object—typically, it will be a GenInteraction, a GenTrigger, or a GenValue.

For example, if you wanted the sample application to have two extra triggers added to the Size menu, you would use ATTR_GEN_CONTROL_APP_UI as shown in Code Display 12-3. This example adds two triggers that turn the bottom GenText object on and off. (Of course, you would not likely put such functionality in the Size menu; this is given for illustration.)

This attribute has no effect on the tools managed by the controller, only on its features. To add application-specific tools to a controller, you must use ATTR_GEN_CONTROL_APP_TOOLBOX_UI.

Code Display 12-3 Adding UI to a Controller

```
/* This display shows the modified PointSizeControl object and the additional
 * UI gadgetry required to add two triggers to it. Although the triggers are
 * shown here in the same resource block as the controller, they do not have to
 * be. They do, however, have to be run by the same thread as the controller. */
```

```
@object PointSizeControlClass PSCTSizeControl = {
    GI_visMoniker = ’z’, “Sizes”;        /* Give the controller a name */
    GII_visibility = GIV_POPUP;          /* Make the controller a menu */
        /* The following attribute specifies the top object of a generic
         * tree to be included with the controller’s UI. This attribute does
         * not affect the toolbar implementation of the controller. */
    ATTR_GEN_CONTROL_APP_UI = (@PSCTSpecialTrigs);/* must be an optr */
}

/* This GenInteraction and its children (the two GenTriggers) will be included in
 * the controller’s default representation (typically a menu). The GenInteraction
 * will appear as a submenu in OSF/Motif. All of the objects must be set not
 * usable (~GS_USABLE); they will be made usable by the controller when it becomes
 * usable. */

@object GenInteractionClass PSCTSpecialTrigs = {
    GI_comp = @PSCTEnableTrig, @PSCTDisableTrig;
    GI_states = @default & ~GS_USABLE;
    GII_visibility = GIV_POPUP;          /* Appear as a submenu-type item */
}

/* The triggers set the bottom GenText object usable or not usable. The instance
 * data of the triggers is unimportant for this example, but it is shown here
 * to complete the example. */

@object GenTriggerClass PSCTEnableTrig = {
    GI_visMoniker = “Use Two Text Fields”;
    GI_states = @default & ~GS_USABLE;
    GTI_actionMsg = MSG_GEN_SET_USABLE;
    GTI_destination = PSCBotTextObj;
    ATTR_GEN_TRIGGER_ACTION_DATA = (VUM_NOW);
}

@object GenTriggerClass PSCTDisableTrig = {
    GI_visMoniker = “Use One Text Field”;
    GI_states = @default & ~GS_USABLE;
    GTI_actionMsg = MSG_GEN_SET_NOT_USABLE;
    GTI_destination = PSCBotTextObj;
    ATTR_GEN_TRIGGER_ACTION_DATA = (VUM_NOW);
}
```

12.3

◆Objects

12.3.2 Using Tools

HINT_GEN_CONTROL_TOOLBOX_ONLY,
ATTR_GEN_CONTROL_REQUIRE_TOOLBOX_UI,
ATTR_GEN_CONTROL_PROHIBIT_TOOLBOX_UI,
ATTR_GEN_CONTROL_APP_TOOLBOX_UI

As stated earlier, any GenControl object can be manifested either as menus and menu items or as a set of tools. In fact, the controller can potentially have both its menu and its tools usable at once. Creating and using a toolbox or tool bar is a bit more complex than simply including the controller, however.

You can add the use of tool bars and tool boxes to your application in two ways: First, you can simply add a GenToolControl object and associated GenToolGroups and let them do all the work for you. Second, you can interact directly with the controller object(s) to put up, take down, and otherwise manage the tools. In nearly all cases, the first is preferable.

This section focuses on the use of a GenToolControl and GenToolGroup objects to manage and place your tools and toolboxes. If you want to manage tools without using a GenToolControl, you will have to know more about **GenControlClass**; see section 12.4 on page 810 for complete details.

12.3.2.1 Components of Tool Management

Tools are most often represented by specific buttons or popup lists presented in various toolbars for easy use. They are functionally redundant to the menu features represented by the normal features of the controller, but the tools and menus can both be visible and usable at the same time.

To provide and manage tools, you need to understand the following components:

◆ Controller
The controller is the same GenControl subclass object discussed and shown in the previous sections. You can set minimum and maximum tool sets just like setting minimum and maximum feature set.

◆ Tool Group
Each controller object should have exactly one corresponding

GenToolGroup object. The GenToolGroup manages the controller's tools and is, in turn, managed by the GenToolControl object (below).

- Tool Controller
  You should have exactly one GenToolControl object; this object is a controller that allows the user to turn on and off individual tools and entire tool sets. It also controls placement of each tool group—which tool bar the tool group appears in.

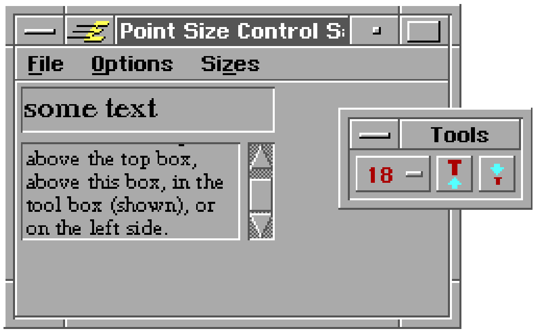
12.3

- Tool Bars and Tool Boxes
  You can have any number of tool bars and floating tool boxes. A tool bar or tool box is simply a GenInteraction set up properly and accessible by the user. Examples of tool bars are given in the next sections.

- Data Tables
  Because the GenToolControl needs to describe both the tool bars and the tool groups to the user, you must set up special data chunks containing tables of names for each tool bar and tool group you use.

12.3.2.2 Using a Basic Tool Box

The most basic tool configuration includes a floating tool box in which the controllers' tools appear. To use a tool box, you only need to define a toolbox GenInteraction, add a GenToolControl object, and put them both in a menu. Typically, this will be an "Options" menu.



some text

above the top box, above this box, in the tool box (shown), or on the left side.

Figure 12-4 *A Tool Box*

The PointSizeControl tools have been placed in the floating tool box. The numbered tool is actually a popup list; the others are triggers.

Objects

Code Display 12-4 shows the configuration required for providing a basic tool box as well as the controller's default menu. (You could prohibit the appearance of the menu by setting ATTR_GEN_CONTROL_TOOLBOX_ONLY in the controller's instance data.) This example provides all the tools by setting the application's UI level in the GenApplication object. All the tools will then appear in the toolbox and its associated popups as shown in Figure 12-4.

Setting Up the Tool Box

The tool box must be a GenInteraction object; for a floating tool box (as opposed to a tool bar), set it to be a dialog box (GIV_DIALOG). It should also have a moniker and HINT_TOOLBOX. The tool box must also be given a name. This name is used by the GenToolControl to identify the tool box. The name is set in a separate chunk and is a simple character string.

At startup, the GenToolGroup for the point size controller is set as a child of the floating tool box. The GenToolControl will move the GenToolGroup to other toolbars if possible; the tools must be set somewhere at startup, though, and the tool box is the logical starting point.

Setting Up the Tool Controller

The GenToolControl object will be of **GenToolControlClass**, as shown. It must at least have the *GTCI_toolboxList* field set to the chunk handle of a table of tool locations. In this example, the only location available to tools is in the floating tool box; other locations may be specified as detailed in “Tool Placement” on page 807 below.

The table must appear in its own chunk in the same resource block as the GenToolControl object. It is an array of **ToolboxInfo** structures, each of which contains two optrs. The first is the optr of a tool location (typically a GenInteraction that may contain tools), and the second is the optr of the associated name. The structure of **ToolboxInfo** is shown below:

```
typedef struct {
      optr  TI_object;  /* A GenInteraction that
                         * can contain tools */
      optr  TI_name;    /* The name chunk of the
                         * TI_object object */
} ToolboxInfo;
```

The table is defined as shown in Code Display 12-4. If you had other controllers to be managed by the tool control, you would add other entries separated by commas.

The GenToolControl also has a tool group list indicating the name of each GenToolGroup. It is set up in a similar form to the tool box table.

12.3 **Code Display 12-4 Providing a Basic Tool Box**

```
/* This code display shows the entire psctext.goc file, with changes noted. All
 * unchanged code has had its comments stripped. For descriptions, see earlier
 * displays in this chapter. */

@include <stdapp.goh>
@include <ui.goh>
@include <Objects/Text/tCtrlC.goh>

@class  PSCTextProcessClass, GenProcessClass;
@endc

@classdecl PSCTextProcessClass, neverSaved;

@start  AppResource;
@object GenApplicationClass PSCTextApp = {
    GI_visMoniker = “Point Size Control Sample Application”;
    GI_comp = @PSCTPrimary;
    gcnList(MANUFACTURER_ID_GEOWORKS, GAGCNLT_WINDOWS) = @PSCTPrimary;
    gcnList(MANUFACTURER_ID_GEOWORKS, GAGCNLT_SELF_LOAD_OPTIONS) =
                                          @PSCTSizeControl, @PSCTToolControl;
}

@end    AppResource

@start  Interface;

/*      GenPrimary Object
 * Typically, an Options menu will be set up as a child of the Primary and
 * the tool box and tool control will be children of that menu. */

@object GenPrimaryClass PSCTPrimary = {
    GI_comp =   @PSCTSizeControl, @PSCTopTextObj, @PSCBotTextObj, @PSCTOptionsMenu;
    HINT_SIZE_WINDOW_AS_DESIRED;
    HINT_ORIENT_CHILDREN_VERTICALLY;
}
```

Objects

```
/*      Options Menu
 * This menu is the parent for both the Tool Box and the tool controller. */
@object GenInteractionClass PSCTOptMenu = {
    GI_comp = @PSCTToolBox, @PSCTToolControl;
    GII_visibility = GIV_POPUP;
    ATTR_GEN_INTERACTION_GROUP_TYPE = (GIGT_OPTIONS_MENU);
}

/*      Tool Box Location Table
 * The Tool Box Location Table is used by the GenToolControl object
 * to associate tool locations with their names. The table is an array
 * of ToolboxInfo structures. Multiple entries would be separated with
 * commas. */

@chunk ToolboxInfo PSCTToolboxList[] = {
    {@PSCTToolBox, @PSCTToolBoxName}     /* The single tool location is the
                                          * floating tool box PSCTToolBox. */
};

/*      Tool Group Information Table
 * The Tool Group Information Table is used by the GenToolControl to associate
 * tool groups with their names. The table is an array of ToolGroupInfo structures.
 * Multiple entries would be separated with commas. */

@chunk ToolGroupInfo PSCTToolGroupTable[] = {
    {@PSCTPointSizeToolGroup}
};

/*      Floating Tool Box
 * The Tool Box object is a GenInteraction dialog box. All toolboxes must
 * have HINT_TOOLBOX and may have any additional geometry hints you
 * deem necessary. Because every controller’s GenToolGroup object must
 * be a child of some tool bar, the PSCTPointSizeToolGroup is set at
 * startup as a child of this floating toolbox. */

@object GenInteractionClass PSCTToolBox = {
    GI_visMoniker = ‘T’, “Tools”;
    GI_comp = @PSCTPointSizeToolGroup;
    GII_visibility = GIV_DIALOG;
    HINT_TOOLBOX;
    HINT_ALLOW_CHILDREN_TO_WRAP;
    HINT_ORIENT_CHILDREN_HORIZONTALLY;
    HINT_FULL_JUSTIFY_CHILDREN_HORIZONTALLY;
}
```

```
/* The Tool Box Name must be specified for the tool control object. It
 * must be a character string, and there must be one for each entry in
 * the Tool Location table. */
@chunk char PSCTToolBoxName[] = “Floating Tool Box”;

/*      GenToolControl Object
 * The Tool Control object must have just the GTCI_toolboxList and
 * GTCI_toolGroupList fields set; these contain lists of tool bars and
 * tool groups along with their names. */
```

12.3

```
@object GenToolControlClass PSCTToolControl = {
    GTCI_toolboxList = @PSCTToolboxList;
    GTCI_toolGroupList = @PSCTToolGroupTable;
    HINT_SAME_CATEGORY_AS_PARENT;
}

/*      PointSizeControl        */

@object PointSizeControlClass PSCTSizeControl = {
    GI_visMoniker = ‘z’, “Sizes”;
    GII_visibility = GIV_POPUP;
}

/*      GenToolGroup Object
 * Each controller object has exactly one GenToolGroup object for managing
 * its tools and for management by the GenToolControl. The Tool Group has
 * a single instance field specifying the controller for which it works. */

@object GenToolGroupClass PSCTPointSizeToolGroup = {
    GTGI_controller = @PSCTSizeControl;
}

/*      GenText Objects         */

@object GenTextClass PSCTopTextObj = {
    GI_attrs = @default | GA_TARGETABLE;
    HINT_DEFAULT_FOCUS;
    HINT_DEFAULT_TARGET;
    ATTR_GEN_TEXT_DEFAULT_CHAR_ATTR = ((VTDS_12 << VTDCA_SIZE_OFFSET) |
                                        VTDF_URW_ROMAN);
}
```

Objects

```
@object GenTextClass PSCBotTextObj = {
 GI_attrs = @default | GA_TARGETABLE;
 ATTR_GEN_TEXT_DEFAULT_CHAR_ATTR =
 ((VTDS_12 << VTDCA_SIZE_OFFSET) | VTDF_URW_ROMAN);
}
@end Interface
```

12.3.2.3 Tool Placement

Tools are movable; this means that controller tools can appear in any GenInteraction you may specify in the Tool Location Table. For example, you may want the user to be able to specify where she or he wants the tools to appear: in the floating tool box, at the top of the primary window, at the left of the display, or between the text objects. To support these locations for the tools, all you need to do is set up empty GenInteraction objects in the appropriate locations and add entries to the Tool Location Table.

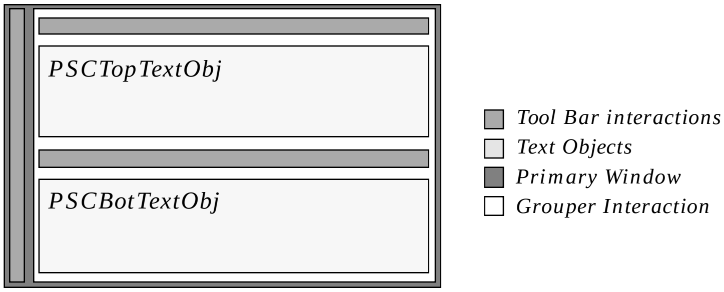


Figure 12-5 *Toolbars of the Sample Application*
Three tool bars in addition to the floating tool box are defined for the new configuration.

The setup described above requires quite a few GenInteraction objects to be added to the application’s generic tree. A line drawing of the geometry with all the empty GenInteractions is given in Figure 12-5, and the new generic tree of the application is shown in Figure 12-6.The code representing this configuration is shown in Code Display 12-5—pay particular attention to the

Tool Location Table and to the fact that each GenInteraction must have the hint HINT_TOOLBOX set in order to receive the tools.

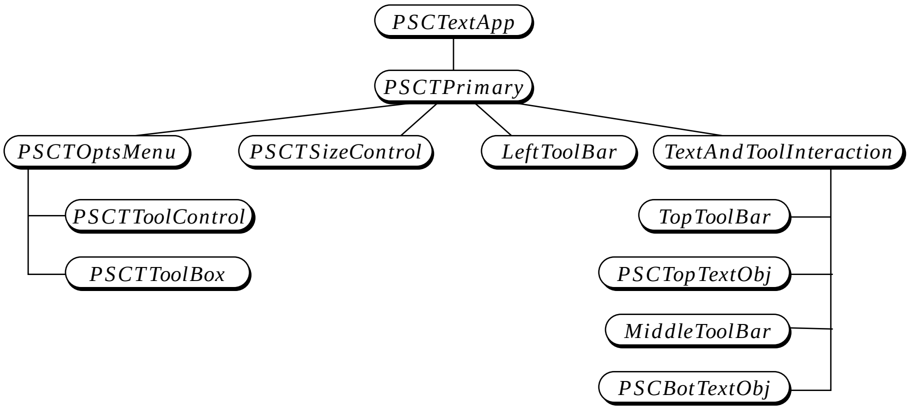


Figure 12-6 *Modified psctext.geo Object Tree*

The Options Menu has two triggers, each of which brings up a dialog box. The TextAndToolInteraction object groups the other objects vertically.

Code Display 12-5 Movable Tools

```
/* This code display builds on Code Display 12-4 on page ◆ 804 to show how tools
 * may be moved around your application’s window by the user. Although this is
 * not difficult to do from scratch, it is quite involved; if you want this
 * functionality, it is best to include a GenToolControl object.
 * This code display only shows those objects that are additional to or altered
 * from the previous display. */

/*      GenPrimary Object
 * Two GenInteractions are made children of the Primary for geometry purposes.
 * The first, LeftToolBar, is actually a tool bar; the second,
 * TextAndToolInteraction, is a grouper interaction for geometry purposes. */

@object GenPrimaryClass PSCTPrimary = {
    GI_comp = @LeftToolBar, @PSCTSizeControl, @PSCTOptionsMenu,
                                    @TextAndToolInteraction;
    gcnList(MANUFACTURER_ID_GEOWORKS, GAGCNLT_SELF_LOAD_OPTIONS) =
                                    @PSCTSizeControl, @PSCTToolControl;
    HINT_SIZE_WINDOW_AS_DESIRED;
    HINT_ORIENT_CHILDREN_HORIZONTALLY;
}
```

◆Objects

```
/*      Tool Location Table
 * The Tool Location Table is updated with the new tool box information. Each of
 * the new tool boxes is given a name, and each must appear in this table.
 * Note that the ToolGroup Information Table does not change. */

@chunk ToolboxInfo PSCTToolboxList[] = {
        {@PSCTToolBox, @PSCTToolBoxName},
        {@LeftToolBar, @LeftToolBarName},
        {@TopToolBar, @TopToolBarName},
        {@MiddleToolBar, @MiddleToolBarName}
};

/*      TextAndToolInteraction Interaction
 * This GenInteraction is used solely as a place holder grouping object to allow
 * the LeftToolBar object to extend the full height of the Primary window. */
@object GenInteractionClass TextAndToolInteraction = {
    GI_comp = @TopToolBar, @PSCTopTextObj, @MiddleToolBar, @PSCBotTextObj;
    HINT_ORIENT_CHILDREN_VERTICALLY;
    HINT_EXPAND_WIDTH_TO_FIT_PARENT;
}

/*      New Tool Box Interactions
 * These GenInteraction objects are all tool boxes that appear in the Tool Location
 * Table. None actually has tools in it on startup; the tool controller allows the
 * user to place the tools of each active controller in any of these tool boxes. */

@object GenInteractionClass LeftToolBar = {
    HINT_TOOLBOX;
    HINT_EXPAND_HEIGHT_TO_FIT_PARENT;
    HINT_ALLOW_CHILDREN_TO_WRAP;
    HINT_ORIENT_CHILDREN_VERTICALLY;
}
@chunk char LeftToolBarName[] = “Left of Text”;

@object GenInteractionClass TopToolBar = {
    HINT_TOOLBOX;
    HINT_EXPAND_WIDTH_TO_FIT_PARENT;
    HINT_ORIENT_CHILDREN_HORIZONTALLY;
    HINT_ALLOW_CHILDREN_TO_WRAP;
}
@chunk char TopToolBarName[] = “Above Text”;
```

```
@object GenInteractionClass MiddleToolBar = {
    HINT_TOOLBOX;
    HINT_EXPAND_WIDTH_TO_FIT_PARENT;
    HINT_ORIENT_CHILDREN_HORIZONTALLY;
    HINT_ALLOW_CHILDREN_TO_WRAP;
}
@chunk char MiddleToolBarName[] = “In Between Text”;
```

12.3.2.4 Adding Application-Specific UI to the Tool Box

ATTR_GEN_CONTROL_APP_TOOLBOX_UI

Occasionally an application may want to add some additional UI gadgetry to a set of controller tools. ATTR_GEN_CONTROL_APP_TOOLBOX_UI is analogous to ATTR_GEN_CONTROL_APP_UI, described in “Adding Application-Specific UI Gadgetry” on page 799. This attribute specifies a generic object tree that can be attached to the controller’s tools as if it were part of the controller normally. For an example of ATTR_GEN_CONTROL_APP_UI’s use, see “Adding Application-Specific UI Gadgetry” on page 799.

12.4 Creating Your Own Controllers

To create your own controller classes, you will have to subclass **GenControlClass** and handle certain messages and data structures. Most applications, however, will find that the controller classes provided with GEOS are adequate for their needs. Some library programmers may want to create their own controller classes.

GenControlClass is a subclass of **GenInteractionClass**. All instance data, hints, and messages appropriate for a GenInteraction are also appropriate for a GenControl object.

Typically, when you create your own controller, you will create a library in which the controller will reside. The controller may be in a library with a particular data object (e.g. **PointSizeControlClass** is part of the text library), or it may be in a library all by itself. For this discussion, the controller is considered to be part of a larger library.

The files of this library are

psCtrl.gp The global parameters file for the controller's library. Only those portions of this file that pertain to the controller are discussed in this section.

psCtrl.goh The header file containing the class definition of the controller class. This is separated from the class' code so the user of the controller can include the **.goh** file and thereby the controller.

psCtrl.goc The code and resource file of the controller class. This file contains the UI objects and the methods of the controller.

All of these files are detailed throughout the following subsections. Each code display explains in which of these files it belongs.

12.4.1 GenControlClass Instance Data

GenControlClass has several instance data fields that determine the controller's features and supported tool set. These instance fields, along with the messages defined for the class, are shown in Code Display 12-6.

Code Display 12-6 GenControlClass Instance Data

```
/* GenControlClass has one static instance data field and several dynamic (vardata)
 * fields. These are shown below. */

        /* The GCI_output field contains the optr of the object to which
         * the controller is currently sending its “apply” messages. This
         * field is typically set to a GenTravelOption (such as TO_APP_TARGET)
         * or a TravelOption (such as TO_OBJ_BLOCK_OUTPUT). */
    @instance   optr    GCI_output;

        /* Controller class objects are set disabled by default. When the
         * controller is initialized, it will set itself enabled. */
        @default        GI_states = (@default & ~GS_ENABLED);

/* the following attributes and hints determine the controller’s
 * feature set. All of these hints and attributes are described in
 * “Using Controllers” on page 796. */
    @vardata void       HINT_GEN_CONTROL_TOOLBOX_ONLY;
    @vardata WordFlags  ATTR_GEN_CONTROL_REQUIRE_UI;
    @vardata WordFlags  ATTR_GEN_CONTROL_PROHIBIT_UI;
    @vardata WordFlags  ATTR_GEN_CONTROL_REQUIRE_TOOLBOX_UI;
    @vardata WordFlags  ATTR_GEN_CONTROL_PROHIBIT_TOOLBOX_UI;

/* The following two attributes control additional UI gadgetry added to a
 * controller object. */
    @vardata optr       ATTR_GEN_CONTROL_APP_UI;
        @reloc  ATTR_GEN_CONTROL_APP_UI, 0, optr;
    @vardata optr       ATTR_GEN_CONTROL_APP_TOOLBOX_UI;
        @reloc  ATTR_GEN_CONTROL_APP_TOOLBOX_UI, 0, optr;

/* The following two hints allow an application to specify the initial
 * state of a controller including its placement, features, and
 * additional UI objects. */
    @vardata GenControlUserData HINT_GEN_CONTROL_MODIFY_INITIAL_UI;
    @vardata GenControlUserData HINT_GEN_CONTROL_USER_MODIFIED_UI;

/* The following temporary data field is used to determine the tool settings
 * for a controller's set of tools. */
    @vardata TempGenControlInstance TEMP_GEN_CONTROL_INSTANCE;
```

In general, a controller class will not have to worry about the instance data fields in Code Display 12-6. Instead, it will handle a particular message, set up some UI resources, and rely on the functionality built into **GenControlClass**.

12.4.2 Subclassing GenControlClass

When creating your own controller, you must subclass **GenControlClass**. You must also follow the steps outlined below (each is described in more detail throughout this section):

1 Define the features and tool records
You must define records and flags for the entire feature set and tool set of the controller. Each bit in one of these records corresponds to a single feature or tool of the controller; by turning these bits on and off, the controller manages which features and tools are available to the user. See "Defining a Controller's Feature and Tool Sets" on page 813.

2 Define the default UI configuration
Define a resource block containing the objects that will be the controller's

default UI representation (typically a menu structure). See “Defining the Default UI and Tool Configurations” on page 815.

3 Define the tool configuration
Define a resource block containing generic UI objects that will make up the controller’s tool set. See “Defining the Default UI and Tool Configurations” on page 815.

4 Handle MSG_GEN_CONTROL_GET_INFO
Every controller *must* handle this message and return critical information about the controller, its features, and its tools. This is the most involved step of creating a controller. See “Mandatory Message Handling” on page 818.

5 Intercept appropriate **GenControlClass** messages
Different controller classes will intercept different **GenControlClass** messages depending on how much additional functionality they require. Most controllers will intercept MSG_GEN_CONTROL_UPDATE_UI.

6 Define and handle controller-specific messages
Some controllers will set up their features and tools so that the messages they generate require some translation. For example, if a controller has one feature that sets a value based on all the other values, that feature may send a “set” message to the controller, which will determine the appropriate value to set. The controller will then pass the result on to the output object.

12.4.2.1 Defining a Controller’s Feature and Tool Sets

Every controller must have a definition of all its features and tools. This definition typically resides in the controller class header file (in this case, **psCtrl.goh**). Applications that use the controller must be able to turn on and off individual tools and features. These definitions take the form of records in which each bit represents a particular feature or tool. No controller may have more than sixteen features or tools; the controller has just one word representing which features and tools are “on.”

To define the feature set of your controller class, define a record type of type **WordFlags** and one flag for each feature. The record type you define may be named anything; typically, however, its name will consist of the acronym of the controller class with the suffix “Features.” For example, the features type and flags of **PointSizeControlClass** are shown below:

```
typedef WordFlags PSCFeatures;
#define PSCF_10         0x0400
#define PSCF_12         0x0200
#define PSCF_14         0x0100
#define PSCF_18         0x0080
#define PSCF_24         0x0040
#define PSCF_36         0x0020
#define PSCF_54         0x0010
#define PSCF_72         0x0008
#define PSCF_SMALLER    0x0004
#define PSCF_LARGER     0x0002
#define PSCF_CUSTOM_SIZE 0x0001
```

Each of the flags represents one feature of the controller. For example, the PSCF_10 flag represents the “10 Point” trigger in the Sizes menu, and the PSCF_SMALLER flag represents the “Smaller” trigger. When a flag is set, the feature it represents is turned on; when clear, its feature is off.

Each controller class must create a similar record type and flags for its tool set. Because the tools are independent of the default features, two different sets of flags must be defined. An example (the **PSCToolboxFeatures** record of **PointSizeControlClass**) follows.

```
typedef WordFlags PSCToolboxFeatures;

#define PSCTF_9         0x0400
#define PSCTF_10        0x0200
#define PSCTF_12        0x0100
#define PSCTF_14        0x0080
#define PSCTF_18        0x0040
#define PSCTF_24        0x0020
#define PSCTF_36        0x0010
#define PSCTF_54        0x0008
#define PSCTF_72        0x0004
#define PSCTF_SMALLER   0x0002
#define PSCTF_LARGER    0x0001
```

After you have defined your controller class’ feature and tool sets, you should define the controller’s default feature and tool sets. The definitions for **PointSizeControlClass** are shown below:

```
#define PSC_DEFAULT_FEATURES  (PSCF_9 | PSCF_10 |
            PSCF_12 | PSCF_14 | PSCF_18 | PSCF_24 |
            PSCF_36 | PSCF_72 | PSCF_CUSTOM_SIZE |
            PSCF_SMALLER | PSCF_LARGER)
#define PSC_DEFAULT_TOOLBOX_FEATURES (PSCTF_9 |
            PSCTF_10 | PSCTF_12 | PSCTF_14 |
            PSCTF_18 | PSCTF_24 | PSCTF_36 |
            PSCTF_72 | PSCTF_SMALLER | PSCTF_LARGER)
```

These values and flags will be used in your handler for the controller message MSG_GEN_CONTROL_GET_INFO.

12.4.2.2 Defining the Default UI and Tool Configurations

After you have determined which features and tools your controller class will support, you must create the UI objects that correspond to them. To do this, declare two separate resource segments—one to contain the feature objects and the other to contain the tool objects. Then declare a third that contains just chunks with text strings in it.

Both resources should be defined *notDetachable*, meaning that the feature and tool objects will not be saved to a state file. In the global parameters file for your controller, each resource must be declared with the *ui-object*, *read-only*, and *shared* flags as below:

```
resource SIZECTRLUI ui-object read-only shared
resource SIZECTRLTOOLUI ui-object read-only shared
resource CONTROLSTRINGS lmem read-only shared
```

The UI resources typically contain list objects and their corresponding items. As an alternative, they can contain triggers and dialogs. These objects are declared as a standard object resource with generic objects, as shown in Code Display 12-7. All these objects must be set not usable (~GS_USABLE).

Code Display 12-7 Controller UI Resources

```
/* This display contains only code that appears in the psCtrl.goc file. The first
 * elements of the file are other included files, followed by a class declaration
 * statement. The two UI resources are shown after that, simplified somewhat;
 * Only redundant objects are left out of the display. */
```

```
/* Include the controller class definition and declare the class structure. */
@include <psCtrl.goh>
@classdecl PointSizeControlClass;

@start  SizeCtrlUI, notDetachable;

/* Define the features UI resource. This resource can contain any objects that may
 * typically appear in a menu (e.g. GenInteractions, GenTriggers, and list
 * objects). This example shows a single list object and a few of its entries. */

@object GenItemGroupClass SizesList = {
    GI_states = @default & ~GS_USABLE;   /* Set the list not usable */
        /* The children of the list are defined below. Each entry in the
         * list will appear as a single menu item. */
    GI_comp =   @Size10Entry, @Size12Entry, @Size14Entry, @Size18Entry,
                @Size24Entry, @Size36Entry, @Size54Entry, @Size72Entry;
        /* The “apply” message will be sent to the destination specified
         * in the GIGI_destination field. */
    GIGI_applyMsg = MSG_PSC_SET_POINT_SIZE_FROM_LIST;
        /* The destination is defined as the TravelOption TO_OBJ_BLOCK_OUTPUT.
         * This will send the apply message to the controller’s output object. */
    GIGI_destination = (TO_OBJ_BLOCK_OUTPUT);
}

/* An example of a GenItem for the above list. all the other children are similar
 * with different monikers and identifiers. The identifiers in this case are
 * equivalent to the point size setting for the feature. */
@object GenItemClass Size10Entry = {
    GI_visMoniker = ‘1’, “1. 10 point”;
    GII_identifier = 10;
}

/* A GenTrigger. Shown below is the “Smaller” menu entry of the Point Size
 * controller. Another trigger (“Larger”) and a GenInteraction (the “Custom
 * Size” entry) are also declared. These objects do not have to be declared
 * as children of any object; they will automatically, like the list above,
 * be designated as children of the controller when it is initialized.
 * Note that all of these objects must also be set not usable. */
@object GenTriggerClass SmallerTrigger = {
    GI_states = @default & ~GS_USABLE;
    GI_visMoniker = ‘S’, “Smaller”;
    GI_kbdAccelerator = control ‘9’;
    GTI_actionMsg = MSG_PSC_SMALLER_POINT_SIZE;
    GTI_destination = (TO_OBJ_BLOCK_OUTPUT);
}

@end    SizeCtrlUI
```

Objects

```
/* Define the Tools UI resource. This follows exactly the same rules as the
 * Features UI resource above, but it represents the UI gadgetry that will appear
 * in the controller’s tool boxes rather than its default menus. */

@start  SizeCtrlToolUI, notDetachable;

@object GenItemGroupClass SizesToolList = {
    /* Same as SizesList above, but with the following hints applied: */
    HINT_ITEM_GROUP_MINIMIZE_SIZE;
    HINT_ITEM_GROUP_DISPLAY_CURRENT_SELECTION;
}

/* The list entry items have the exact configuration as above but different
 * names that reflect their tool usage.
 * The only objects allowed as tools for the Point Size controller are the
 * point size list entries, the larger trigger, and the smaller trigger. The
 * “Custom Size” entry is not allowed in the tool box as a matter of style. */

@object GenTriggerClass SmallerToolTrigger = {
    GI_states = @default & ~GS_USABLE;
    GI_visMoniker = “S”;
        /* The moniker of a tool is typically a graphic. The moniker
         * specified here is text for simplicity. */
    GTI_actionMsg = MSG_PSC_SMALLER_POINT_SIZE;
    GTI_destination = TO_OBJ_BLOCK_OUTPUT;
}

@end    SizeCtrlToolUI

@start  ControlStrings;

/* In addition to the above two resources, you must also create a third that
 * contains name strings for the various tools and features. These name strings
 * will be used by the GenToolControl to identify the feature type in its
 * dialog box. */
@chunk char     PSCName[] = “Point Size”;
@chunk char     Size10Name[] = “10 Point”;
@chunk char     Size12Name[] = “12 Point”;
    /* The rest of the point sizes are similar */
@chunk char     SmallerName[] = “Smaller Point Size”;
@chunk char     LargerName[] = “Larger Point Size”;
@chunk char     CustomSizeName[] = “Custom Point Size”;

@end    ControlStrings
```

12.4.2.3 Mandatory Message Handling

MSG_GEN_CONTROL_GET_INFO, GenControlBuildInfo

12.4

Every controller must handle MSG_GEN_CONTROL_GET_INFO. This message is sent to the controller in several circumstances, and it must return critical information about the controller's state and configuration. It takes a pointer to an empty **GenControlBuildInfo** structure and must fill in all the structure's fields before returning. This structure is shown in Code Display 12-9 with a description of each of its fields following.

Code Display 12-8 The GenControlBuildInfo Structure

```
/* This structure must be filled and returned by the controller class. It details
 * general information as well as specific information about the controller, the
 * controller's features, and the controller's tools. */

typedef struct {
    GenControlBuildFlags        GCBI_flags;
    const char                  *GCBI_initFileKey;
    const GCNListType           *GCBI_gcnList;
    word                        GCBI_gcnCount;
    const NotificationType      *GCBI_notificationList;
    word                        GCBI_notificationCount;
    optr                        GCBI_controllerName;

    MemHandle                   GCBI_dupBlock;
    const GenControlChildInfo   *GCBI_childList;
    word                        GCBI_childCount;
    const GenControlFeaturesInfo *GCBI_featuresList;
    word                        GCBI_featuresCount;
    WordFlags                   GCBI_features;

    MemHandle                   GCBI_toolBlock;
    const GenControlChildInfo   *GCBI_toolList;
    word                        GCBI_toolCount;
    const GenControlFeaturesInfo *GCBI_toolFeaturesList;
    word                        GCBI_toolFeaturesCount;
    WordFlags                   GCBI_toolFeatures;

    char                        *GCBI_helpContext;
    byte                        GCBI_reserved[8];
} GenControlBuildInfo;
```

The following fields define general information about the controller.

GCBI_flags A record of **GenControlBuildFlags**. These flags affect several UI-related and object storage related functions, and they are detailed on page 822.

GCBI_initFileKey
A pointer to a text string indicating the controller's key in the GEOS.INI file. Controller options will be saved under this key.

GCBI_gcnList
A pointer to a list of GCN list types. Objects of this controller class will be added to these GCN lists and will receive notification from them. GCN lists are detailed in "General Change Notification," Chapter 9 of the Concepts Book.

GCBI_gcnCount
The size of the list pointed to by *GCBI_gcnList* above. This size should be *sizeof(GCNListType)* times the number of lists specified.

GCBI_notificationList
A pointer to a list of notification types supported by the controller.

GCBI_notificationCount
The size of the list pointed to by *GCBI_notificationList* above.

GCBI_controllerName
The optr of a chunk containing the text string that serves as the controller's name. This name string is displayed by the GenToolControl in its dialog box to identify the controller.

The following fields define information about the controller's features. These fields will be filled dependent on the features set in the object's instance data and the UI level of the controller.

GCBI_dupBlock
The handle of the resource block containing the controller's feature generic objects. In the example, this would contain the handle of the *SizeCtrlUI* resource.

GCBI_childList
A pointer to a list of **GenControlChildInfo** structures; each of these structures details which features are set and which

should always be set for each of the controller's children. This structure is shown below:

```
typedef struct {
    ChunkHandle   GCCI_object;
    WordFlags     GCCI_featureMask;
    GenControlChildFlags GCCI_flags;
} GenControlChildInfo;
```

Each structure contains the chunk handle of the given child in the resource block, a feature mask indicating which features are possibly supported by the child, and a record indicating whether the child is a feature or not and whether the child is always added to the controller's UI. More specific information is shown on page 823.

GCBI_childCount
The number of children specified in *GCBI_childList* above.

GCBI_featuresList
A pointer to a list of **GenControlFeaturesInfo** structures, one for each child. These structures define the following:

```
typedef struct {
    ChunkHandle   GCFI_object;
    optr          GCFI_name;
    GenControlFeatureFlags GCFI_flags;
} GenControlFeaturesInfo;
```

The three fields are the chunk handle of the child; the optr of the child's name string, as defined in the name string resource; and a record of **GenControlFeatureFlags**. This structure and its fields are described more fully on page 824.

GCBI_featuresCount
The number of **GenControlFeaturesInfo** structures listed in *GCBI_featuresList* above.

GCBI_features
A features mask describing the features supported by the current UI level as specified in the GenApplication's *GAI_appFeatures* field.

The following fields describe information about the controller's tools and their configuration.

GCBI_toolBlock

The handle of the resource block containing the controller's tool generic objects. In the example, this would contain the handle of the *SizeCtrlToolUI* resource.

GCBI_toolList

A pointer to a list of **GenControlChildInfo** structures; each of these structures details which tools are set and which should always be set for each of the controller's children. This structure is shown below:

```
typedef struct {
    ChunkHandle   GCCI_object;
    WordFlags     GCCI_featureMask;
    GenControlChildFlags GCCI_flags;
} GenControlChildInfo;
```

Each structure contains the chunk handle of the given child in the resource block, a feature mask indicating which tools are possibly supported by the child, and a record indicating whether the child is a tool or not (e.g. a list) and whether the child is always added to the controller's tool box UI. More specific information is shown on page 823.

GCBI_toolCount

The number of children specified in *GCBI_toolList* above.

GCBI_toolFeaturesList

A pointer to a list of **GenControlFeaturesInfo** structures, one for each child. These structures define the following:

```
typedef struct {
    ChunkHandle   GCFI_object;
    optr          GCFI_name;
    byte          GCFI_flags;
                  /* GenControlFeatureFlags */
} GenControlFeaturesInfo;
```

The three fields are the chunk handle of the child; the optr of the child's name string, as defined in the name string resource; and a record of **GenControlFeatureFlags**. This structure and its fields are described more fully on page 824.

GCBI_toolFeaturesCount
The number of **GenControlFeaturesInfo** structures in the list pointed to by *GCBI_toolFeaturesList* above.

GCBI_toolFeatures
A tools mask describing the tools supported for the UI level specified in the GenApplication's *GAI_appFeatures* field.

The following field is used by controllers that offer their own help files and help text.

GCBI_helpContext
A pointer to a character string giving the name of the controller's help context. If this is a non-null pointer, then the controller will automatically add ATTR_GEN_HELP_CONTEXT to itself with the specified string.

The structure also has eight bytes that are reserved for the use of GenControlClass, in the *GCBI_reserved* field.

GenControlBuildFlags

This flags record defines several UI-related things about the controller object. Set them appropriate to your controller. The flags are

GCBF_SUSPEND_ON_APPLY
Causes MSG_META_SUSPEND to be sent on feature activation and MSG_META_UNSUSPEND afterward. This is often set by controllers.

GCBF_USE_GEN_DESTROY
Ensures that unused objects can not be freed with **LMemFree()**. Not often set by controllers.

GCBF_SPECIFIC_UI
Indicates that the controller may be implemented in the specific UI and therefore some special action must be taken. Very rarely set by controllers.

GCBF_CUSTOM_ENABLE_DISABLE
Indicates that the controller uses a custom enable/disable mechanism rather than responding to GCN notifications.

GCBF_ALWAYS_UPDATE
Indicates that the controller should always undergo visual

updates even if it appears unnecessary. Not often set by controllers.

GCBF_EXPAND_TOOL_WIDTH_TO_FIT_PARENT
Indicates that the tool width should be expanded to take full advantage of all the space available in the parent composite.

GCBF_ALWAYS_INTERACTABLE
Indicates that the controller should always be on its appropriate GCN lists, even if no part of it is visible. This flag requires GCBF_IS_ON_ACTIVE_LIST.

GCBF_ALWAYS_ON_GCN_LIST
Indicates that the controller should constantly be on the GCN lists rather than periodically adding and removing itself as is done in some optimization code. Not often set by controllers. This flag requires GCBF_IS_ON_ACTIVE_LIST.

GCBF_MANUALLY_REMOVE_FROM_ACTIVE_LIST
Indicates that the controller should not remove itself from the active list in its default detach handler.

GCBF_IS_ON_ACTIVE_LIST
Indicates that the controller is on the active list in its **.goh** file definition.

GCBF_IS_ON_START_LOAD_OPTIONS_LIST
Indicates this controller must be on the startup load options list.

GCBF_NOT_REQUIRED_TO_BE_ON_SELF_LOAD_OPTIONS_LIST
Indicates this controller is not required to be on any options-load list.

GenControlChildInfo

The **GenControlChildInfo** structure defines the features or tools appropriate to each object in a controller's UI resources. It has the following structure, and its fields are described below:

```
typedef struct {
      ChunkHandle             GCCI_object;
      WordFlags               GCCI_featureMask;
      GenControlChildFlags    GCCI_flags;
} GenControlChildInfo;
```

GCCI_object The chunk handle of the object in the appropriate resource.

GCCI_featureMask
The feature mask representing the feature set represented by the object.

GCCI_flags A record of **GenControlChildFlags**, the flags of which are described below.

The **GenControlChildFlags** flags are

GCCF_NOTIFY_WHEN_ADDING
This flag indicates that the child will be notified before the feature is added and set usable with MSG_GEN_CONTROL_NOTIFY_ADDING_FEATURE.

GCCF_ALWAYS_ADD
This flag indicates that the child object should always be added to the controller's UI, even if it is not specified.

GCCF_IS_DIRECTLY_A_FEATURE
This flag indicates that the child is a feature in itself and thus on the feature list. This is typically set for most objects. (It is not set, for example, for list objects, whose children are the actual features.)

GenControlFeaturesInfo

The **GenControlFeaturesInfo** structure describes each UI feature's name and certain flags. The structure is defined below, and its fields are described following:

```
typedef struct {
      ChunkHandle             GCFI_object;
      optr                    GCFI_name;
      GenControlFeatureFlags  GCFI_flags;
} GenControlFeaturesInfo;
```

GCFI_object The chunk handle of the child in the appropriate resource block.

GCFI_name The optr of the chunk containing the object's name. This name is used by the GenToolControl to represent the particular feature or tool in its dialog box.

GCFI_flags A record of **GenControlFeatureFlags**, reserved.

■ MSG_GEN_CONTROL_GET_INFO

```
void        MSG_GEN_CONTROL_GET_INFO(
            GenControlBuildInfo *info);
```

This message must be handled by all controllers. It takes an empty **GenControlBuildInfo** structure and fills it; this message is called in several circumstances by different objects and controller methods.

Source: Unrestricted—typically generated in **GenControlClass** methods.

Destination: Any controller object.

Parameters: *info* A pointer to an empty **GenControlBuildInfo** structure.

Return: The **GenControlBuildInfo** structure filled with the appropriate controller information.

Interception: Every controller class *must* intercept this message. There is no need to call the superclass anywhere in the handler.

Code Display 12-9 MSG_GEN_CONTROL_GET_INFO Handler

```
/* This method is a sample of how to handle MSG_GEN_CONTROL_GET_INFO. It is
 * specific to UICTextStyleControlClass. To handle this message, it is easiest to
 * set up a number of static local variables with the base information and set
 * the structure to these variables. */

/* Handler for MSG_GEN_CONTROL_GET_INFO
 * void (GenControlBuildInfo *info);     */

@method UICTextStyleControlClass, MSG_GEN_CONTROL_GET_INFO {

        /* General information constants */

 static const char TSC_IniFileKey[] = “textStyleControl”;

 static const GCNListType TSC_gcnList[] = {
        {MANUFACTURER_ID_GEOWORKS, GAGCNLT_APP_TARGET_NOTIFY_TEXT_CHAR_ATTR_CHANGE}
 };

 static const NotificationType TSC_notifyTypeList[] = {
        {MANUFACTURER_ID_GEOWORKS, GWNT_TEXT_CHAR_ATTR_CHANGE}
 };

        /* Features information constants */
```

```
static const GenControlChildInfo TSC_childList[] = {
    {@PlainTextList, TSCF_PLAIN, GCCF_IS_DIRECTLY_A_FEATURE},
    {@TextStyleList, TSCF_BOLD|TSCF_ITALIC|TSCF_UNDERLINE|TSCF_STRIKE_THRU|
     TSCF_SUBSCRIPT|TSCF_SUPERSCRIPT, 0}

};

    /* The order of this list is actually backwards from the
     * record it reflects. */
static const GenControlFeaturesInfo TSC_featuresList[] = {
     {@SuperscriptEntry, @SuperscriptName, 0},
    {@SubscriptEntry, @SubscriptName, 0},
    {@StrikeThruEntry, @StrikeThruName, 0},
    {@UnderlineEntry, @UnderlineName, 0},
    {@ItalicEntry, @ItalicName, 0},
    {@BoldEntry, @BoldName, 0},
    {@PlainTextList, @PlainTextName, 0}
};

    /* Tools information constants */

static const GenControlChildInfo TSC_toolList[] = {
    {@PlainTextToolList, TSCTF_PLAIN, GCCF_IS_DIRECTLY_A_FEATURE},
    {@TextStyleToolList, TSCTF_BOLD|TSCTF_ITALIC|TSCTF_UNDERLINE|
     TSCTF_STRIKE_THRU|TSCTF_SUBSCRIPT|TSCTF_SUPERSCRIPT, 0} };
};

static const GenControlFeaturesInfo TSC_toolFeaturesList[] = {
    {@SuperscriptToolEntry, @SuperscriptName, 0},
    {@SubscriptToolEntry, @SubscriptName, 0},
    {@StrikeThruToolEntry, @StrikeThruName, 0},
    {@UnderlineToolEntry, @UnderlineName, 0},
    {@ItalicToolEntry, @ItalicName, 0},
    {@BoldToolEntry, @BoldName, 0},
    {@PlainTextToolList, @PlainTextName, 0}
};

    /* Our constant for the GenControlBuildInfo structure.
     * Fields with a marker to the left of their names are
     * filled in dynamically by the handler following the
     * constant definition. */
static const GenControlBuildInfo TSC_dupInfo = {
    GCBF_SUSPEND_ON_APPLY,                  /* GCBI_flags */
    TSC_IniFileKey,                         /* GCBI_initFileKey */
    TSC_gcnList,                            /* GCBI_gcnList */
    ARRAY_LEN(TSC_gcnList,GCNListType),     /* GCBI_gcnCount */
    TSC_notifyTypeList,                     /* GCBI_notificationList */
    ARRAY_LEN(TSC_notifyTypeList, NotificationType),
                                                 /* GCBI_notificationCount */
        @TSCName,                                /* GCBI_controllerName */

        /* ## */ NullHandle,                     /* GCBI_dupBlock */
        TSC_childList,                           /* GCBI_childList */
        ARRAY_LEN(TSC_childList, GenControlChildInfo),
                                                 /* GCBI_childCount */
        TSC_featuresList,                        /* GCBI_featuresList */
        ARRAY_LEN(TSC_featuresList, GenControlFeaturesInfo),
                                                 /* GCBI_featuresCount */
        TSC_DEFAULT_FEATURES,                    /* GCBI_features */
        /* ## */ NullHandle,                     /* GCBI_toolBlock */
        TSC_toolList,                            /* GCBI_toolList */
        ARRAY_LEN(TSC_toolList, GenControlChildInfo),
                                                 /* GCBI_toolCount */
        TSC_toolFeaturesList,                    /* GCBI_toolFeaturesList */
        ARRAY_LEN(TSC_toolFeaturesList, GenControlFeaturesInfo),
                                                 /* GCBI_toolFeaturesCount */
        TSC_DEFAULT_TOOLBOX_FEATURES             /* GCBI_toolFeatures */
    };

        /* Here is the code that fills in the above missing fields and
         * returns the proper structure. */

    /* Copy the structure containing most of the correct information. */
    memcpy(info, MemLockFixedOrMovable(&TSC_dupInfo), sizeof(GenControlBuildInfo));
    MemUnlockFixedOrMovable(&TSC_dupInfo);

    /* Fill the remaining fields in manually. */
    info->GCBI_dupBlock = HandleOf(@PlainTextList);
    info->GCBI_toolBlock = HandleOf(@PlainTextToolList);
}
```

12.4.3 Advanced GenControlClass Usage

Everything you need to create a basic custom controller class is detailed in the previous sections. **GenControlClass**, however, has a number of messages and structures that will be used by some subclasses, though this will be relatively rare. This section details these messages and structures.

12.4.3.1 Adding and Removing UI Gadgetry

```
MSG_GEN_CONTROL_GENERATE_UI, MSG_GEN_CONTROL_DESTROY_UI,
MSG_GEN_CONTROL_GENERATE_TOOLBOX_UI,
MSG_GEN_CONTROL_DESTROY_TOOLBOX_UI,
MSG_GEN_CONTROL_UPDATE_UI,
MSG_GEN_CONTROL_ENABLE_DISABLE,
MSG_GEN_CONTROL_ADD_APP_UI,
MSG_GEN_CONTROL_ADD_APP_TOOLBOX_UI,
MSG_GEN_CONTROL_REBUILD_NORMAL_UI,
MSG_GEN_CONTROL_REBUILD_TOOLBOX_UI
```

GenControlClass has several messages that add, remove, and manipulate the controller UI gadgetry. Keep in mind also that, as a subclass of **GenInteractionClass** and thereby of **GenClass**, the GenControl also can use any of the generic UI messages for object tree manipulation. These messages are detailed in the following list.

■ MSG_GEN_CONTROL_GENERATE_UI

```
void    MSG_GEN_CONTROL_GENERATE_UI();
```

This message generates the UI gadgetry for the controller.

Source: Unrestricted—sent by self as part of specific UI build.

Destination: Any GenControl object

Interception: If subclassed to add functionality, the subclass must call the superclass at the beginning of the handler.

Warnings: If you intercept this message, you should also intercept MSG_GEN_CONTROL_DESTROY_UI, below.

■ MSG_GEN_CONTROL_DESTROY_UI

```
void    MSG_GEN_CONTROL_DESTROY_UI();
```

This message destroys the UI gadgetry for the controller.

Source: Unrestricted—sent by self when being taken off the screen.

Destination: Any GenControl object

Interception: Any subclass that intercepts MSG_GEN_CONTROL_GENERATE_UI must intercept this. The subclass must call the superclass at the end of the handler.

■ MSG_GEN_CONTROL_GENERATE_TOOLBOX_UI

```
void    MSG_GEN_CONTROL_GENERATE_TOOLBOX_UI(
        optr   parent);
```

This message generates the UI gadgetry for the controller's tool box.

Source: Sent by tool box object via the active list as part of its specific UI build mechanism.

Destination: The GenControl object that owns the tool box.

Parameters: *parent* The optr of GenInteraction that will be the parent of the controller's tools.

Return: Nothing.

Interception: A subclass may intercept this to add UI gadgetry to the toolbox when the toolbox is built. The subclass must call the superclass at the beginning of the handler.

Warnings: Any controller that subclasses this message must also subclass MSG_GEN_CONTROL_DESTROY_TOOLBOX_UI as well.

■ MSG_GEN_CONTROL_DESTROY_TOOLBOX_UI

```
void    MSG_GEN_CONTROL_DESTROY_TOOLBOX_UI();
```

This message destroys all toolbox UI associated with the controller.

Source: Sent by the tool box being destroyed as part of its specific UI destruction mechanism.

Destination: The GenControl object that owns the tool box.

Interception: Any controller class that subclasses the message MSG_GEN_CONTROL_GENERATE_TOOLBOX_UI must also subclass this message. The subclass must call its superclass at the end of its handler.

■ MSG_GEN_CONTROL_UPDATE_UI

```
void        MSG_GEN_CONTROL_UPDATE_UI(@stack
            MemHandle          toolBlock,
            MemHandle          childBlock,
            WordFlags          toolboxFeatures,
            WordFlags          features,
            MemHandle          data,
            word               changeID,
            ManufacturerID     manufID);
```

12.4

This message updates all UI components for the recipient controller.

Source: Sent by **GenControlClass** in its default handler for MSG_META_NOTIFY_WITH_DATA_BLOCK.

Destination: The GenControl object being updated.

Parameters: *toolBlock* Handle of the object resource block containing the controller's tool UI gadgetry.

childBlock Handle of the object resource block containing the controller's default UI gadgetry. This may be taken from the TEMP_GEN_CONTROL_INSTANCE vardata field.

toolboxFeatures A record of flags indicating which tools are currently on for the controller. This may be taken from the TEMP_GEN_CONTROL_INSTANCE vardata field.

features A record of flags indicating which features are currently on for the controller. This may be taken from the TEMP_GEN_CONTROL_INSTANCE vardata field.

data The data block handle passed with the notification message.

changeID The type of update to undergo, as passed with the notification message.

manufID The manufacturer ID of the notification type passed with the notification message.

Interception: Controllers should intercept this message in order to properly update their UI gadgetry. There is no default handler for this message.

◆ Objects

Structures: The TEMP_GEN_CONTROL_INSTANCE vardata field is of type **TempGenControlInstance**, which has the following structure:

```
typedef struct {
      GenControlInteractableFlags
                        TGCI_interactableFlags;
      MemHandle         TGCI_childBlock;
      MemHandle         TGCI_toolBlock;
      optr              TGCI_toolParent;
      WordFlags         TGCI_features;
      WordFlags         TGCI_toolboxFeatures;
      GCNListType       TGCI_activeNotificationList;
      GenControlInteractableFlags TGCI_upToDate;
} TempGenControlInstance;
```

TGCI_interactableFlags
This record describes which, if any, of a controller's UI parts is interactable (visible and usable). Its flags are listed below.

TGCI_childBlock
The handle of the resource block containing the controller's default UI objects.

TGCI_toolBlock
The handle of the resource block containing the tool objects.

TGCI_toolParent
The optr of the object passed with MSG_GEN_CONTROL_GENERATE_TOOLBOX_UI, if any (tools will be added to this object).

TGCI_features
A record of flags indicating which controller features are currently active.

TGCI_toolboxFeatures
A record of flags indicating which controller tools are currently active.

TGCI_activeNotificationList
The notification type currently active.

TGCI_upToDate
The status of **GenControlInteractableFlags** at the time of the last notification; that is, which portions of the controller's

UI were up to date. This is used by the specific UI for drawing optimizations.

The flags for the **GenControlInteractableFlags** record are listed below:

GCIF_CONTROLLER
This flag indicates the controller object is interactable.

GCIF_TOOLBOX_UI
This flag indicates that some or all of the controller's tool gadgetry is interactable.

GCIF_NORMAL_UI
This flag indicates that some or all of the controller's normal feature gadgetry is interactable.

■ MSG_GEN_CONTROL_ENABLE_DISABLE

```
void        MSG_GEN_CONTROL_ENABLE_DISABLE(
            Message             msg;
            VisUpdateMode       updateMode);
```

This message enables or disables the controller object as well as its default and tool box UI gadgetry.

Source: Sent by **GenControlClass** to the controller object when it receives MSG_META_NOTIFY_WITH_DATA_BLOCK.

Destination: Sent by controller to itself.

Parameters: *msg* Either MSG_GEN_SET_ENABLED or MSG_GEN_SET_NOT_ENABLED as appropriate.

updateMode A **VisUpdateMode** indicating when the visual update should occur.

Interception: Unlikely—typically should not be intercepted.

■ MSG_GEN_CONTROL_ADD_APP_UI

```
void        MSG_GEN_CONTROL_ADD_APP_UI(
            optr   appUI);
```

This message adds the passed UI object to the controller's generic tree as if it had been originally defined in the default child block. By default, the new object is added as the last child of the controller.

Source: Unrestricted—generated as part of the default functionality of MSG_GEN_CONTROL_GENERATE_UI if the controller has ATTR_GEN_CONTROL_APP_UI set.

Destination: Any GenControl object.

Parameters: *appUI* The optr of the object to be added.

Interception: Should be intercepted by controllers that wish to add the new object somewhere other than as the last child.

■ MSG_GEN_CONTROL_ADD_APP_TOOLBOX_UI

```
void        MSG_GEN_CONTROL_ADD_APP_TOOLBOX_UI(
            optr   appUI);
```

This message adds the passed UI object to the controller's tool UI gadgetry as if it had been defined as a tool in the tool resource block. By default, the new object is added as the last child of the controller.

Source: Unrestricted—generated as part of the default functionality of MSG_GEN_CONTROL_GENERATE_TOOLBOX_UI if the controller has ATTR_GEN_CONTROL_APP_TOOLBOX_UI set.

Destination: Any GenControl object.

Parameters: *appUI* The optr of the object to be added.

Interception: Should be intercepted by controllers that wish to add the new object somewhere other than as the last child.

■ MSG_GEN_CONTROL_REBUILD_NORMAL_UI

```
void        MSG_GEN_CONTROL_REBUILD_NORMAL_UI();
```

This message forces the normal UI of the controller to be rebuilt; each component will be visually destroyed and rebuilt.

Source: Unrestricted.

Destination: Any GenControl object.

Interception: Should not be intercepted.

■ MSG_GEN_CONTROL_REBUILD_TOOLBOX_UI

```
void        MSG_GEN_CONTROL_REBUILD_TOOLBOX_UI();
```

This message forces the toolbox UI of the controller to be rebuilt; each component will be visually destroyed and rebuilt.

Source: Unrestricted.

Destination: Any GenControl object.

Interception: Should not be intercepted.

12.4.3.2 Adding and Removing Features and Tools

```
MSG_GEN_CONTROL_SCAN_FEATURE_HINTS,
MSG_GEN_CONTROL_ADD_FEATURE,
MSG_GEN_CONTROL_REMOVE_FEATURE,
MSG_GEN_CONTROL_ADD_TOOLBOX_FEATURE,
MSG_GEN_CONTROL_REMOVE_TOOLBOX_FEATURE,
MSG_GEN_CONTROL_GET_NORMAL_FEATURES,
MSG_GEN_CONTROL_GET_TOOLBOX_FEATURES
```

Besides being able to set and clear features of a given controller, both the controller class and outside agents can dynamically alter the features of a given controller object. **GenControlClass** has several messages that you can use for this purpose; these messages are detailed below.

■ MSG_GEN_CONTROL_SCAN_FEATURE_HINTS

```
void    MSG_GEN_CONTROL_SCAN_FEATURE_HINTS(
        GenControlUIType    type,
        GenControlScanInfo  *info);
```

This message scans the feature hints set for the controller object to set the required and prohibited features.

Source: Unrestricted—Sent by **GenControlClass** to itself in numerous circumstances.

Destination: The GenControl object to be scanned.

Parameters: *type* An indicator whether the normal or tool box UI hints are to be scanned. This should be GCUIT_NORMAL for the normal hints, GCUIT_TOOLBOX for the tool box hints.

info A pointer to an empty **GenControlScanInfo** structure to be filled in by the handler. This structure is shown below.

Return: No value is returned directly.

info The pointer to the **GenControlScanInfo** structure filled in by the method.

Interception: Should not be intercepted.

Structures: The **GenControlScanInfo** structure has the following fields:

```
typedef struct {
      WordFlags   GCSI_userAdded;
      WordFlags   GCSI_userRemoved;
      WordFlags   GCSI_appRequired;
      WordFlags   GCSI_appProhibited;
} GenControlScanInfo;
```

GCSI_userAdded
A record of features added by the user through the use of the GenToolControl object. This record also reflects the use of MSG_GEN_CONTROL_ADD_FEATURE.

GCSI_userRemoved
A record of features removed by the user through the use of the GenToolControl object.

GCSI_appRequired
A record of features required; this is set for the controller with ATTR_GEN_CONTROL_REQUIRE_TOOLBOX_UI or ATTR_GEN_CONTROL_REQUIRE_UI.

GCSI_appProhibited
A record of features prohibited; this is set for the controller with ATTR_GEN_CONTROL_PROHIBIT_TOOLBOX_UI or ATTR_GEN_CONTROL_PROHIBIT_UI.

■ MSG_GEN_CONTROL_ADD_FEATURE

```
void    MSG_GEN_CONTROL_ADD_FEATURE(
        WordFlags featureToAdd);
```

This message adds a feature to those currently supported by the controller. This is used for the default settings only, not for tool box UI. The controller is destroyed and then rebuilt with the new feature, causing it to be marked for saving to the state file in the new configuration.

Source: Unrestricted.

Destination: Any GenControl object.

Parameters: *featureToAdd* A feature record with the flag of the feature to be added set.

Interception: Should not be intercepted.

■ MSG_GEN_CONTROL_REMOVE_FEATURE

```
void        MSG_GEN_CONTROL_REMOVE_FEATURE
            WordFlags featureToRemove);
```

This message removes a feature from those currently supported by the controller. It does not affect the active tools. The controller will be marked for saving to the state file in the new configuration; it is destroyed and rebuilt with feature removed.

Source: Unrestricted.

Destination: Any GenControl object.

Parameters: *featureToRemove* A feature record with the flag of the feature to be removed set.

Interception: Should not be intercepted.

■ MSG_GEN_CONTROL_ADD_TOOLBOX_FEATURE

```
void        MSG_GEN_CONTROL_ADD_TOOLBOX_FEATURE(
            WordFlags featureToAdd);
```

This message adds a tool to those currently supported by the controller. This is used for the tool box settings only, not for the default UI features. The controller is destroyed and then rebuilt with the new tool, causing it to be marked for saving to the state file in the new configuration.

Source: Unrestricted.

Destination: Any GenControl object.

Parameters: *featureToAdd* A feature record with the flag of the to be added set.

Interception: Should not be intercepted.

■ MSG_GEN_CONTROL_REMOVE_TOOLBOX_FEATURE

```
void        MSG_GEN_CONTROL_REMOVE_TOOLBOX_FEATURE(
            WordFlags featureToRemove);
```

This message removes a tool from those currently supported by the controller. It does not affect the default active feature list. The controller will be marked for saving to the state file in the new configuration; it is destroyed and rebuilt with the tool removed.

Source: Unrestricted.

Destination: Any GenControl object.

Parameters: *featureToRemove* A feature record with the flag of the tool to be removed set.

Interception: Should not be intercepted.

■ MSG_GEN_CONTROL_GET_NORMAL_FEATURES

```
void MSG_GEN_CONTROL_GET_NORMAL_FEATURES(
        GenControlGetFeaturesReturn *return);
```

This message returns a structure indicating which of the default features of the controller are currently active, which are required, and which are prohibited.

Source: Unrestricted—typically GenToolControl.

Destination: Any GenControl object.

Parameters: *return* A pointer to an empty structure to be returned.

Return: The *return* parameter, upon return, points to a filled **GenControlGetFeaturesReturn** structure.

Structures: The **GenControlGetFeaturesReturn** structure is shown below:

```
typedef struct {
      WordFlags   GCSR_features;
      WordFlags   GCSR_required;
      WordFlags   GCSR_prohibited;
      WordFlags   GCSR_supported;
} GenControlGetFeaturesReturn;
```

GCSR_features
A record of flags indicating which features are currently active.

GCSR_required
A record of flags indicating which features, at the least, are required for the controller.

GCSR_prohibited
A record of flags indicating which features are prohibited to the controller.

GCSR_supported
A record of flags indicating the full range of supported features.

■ MSG_GEN_CONTROL_GET_TOOLBOX_FEATURES

```
void MSG_GEN_CONTROL_GET_TOOLBOX_FEATURES(
        GenControlGetFeaturesReturn *return);
```

This message returns a structure indicating which of the controller's tools are currently active, which are required, and which are prohibited.

Source: Unrestricted—typically GenToolControl.

Destination: Any GenControl object.

Parameters: *return* A pointer to an empty structure to be returned.

Return: The *return* parameter, upon return, points to a filled **GenControlGetFeaturesReturn** structure.

Structures: The **GenControlGetFeaturesReturn** structure is shown in MSG_GEN_CONTROL_GET_NORMAL_FEATURES, above.

12.4.3.3 Working with GCN Lists

MSG_GEN_CONTROL_ADD_TO_GCN_LISTS, MSG_GEN_CONTROL_REMOVE_FROM_GCN_LISTS

Because **GenControlClass** uses GCN lists, it has two messages that add itself to and remove itself from the lists. You can intercept or send these if you need to; applications generally will not, however. These two messages are detailed below.

■ MSG_GEN_CONTROL_ADD_TO_GCN_LISTS

```
void    MSG_GEN_CONTROL_ADD_TO_GCN_LISTS();
```

This message adds the controller object to the GCN lists specified by its return value of MSG_GEN_CONTROL_GET_INFO. It will force a status update to ensure that the controller updates itself.

Source: Generated internally on startup and at other times.

Destination: Sent to self.

Interception: Generally not intercepted. Subclasses should use this only to see when the controller is being added to its lists; it should call the superclass to do the actual addition.

■ MSG_GEN_CONTROL_REMOVE_FROM_GCN_LISTS

```
void        MSG_GEN_CONTROL_REMOVE_FROM_GCN_LISTS();
```

This message removes the controller from all its GCN lists.

Source: Generated internally on shutdown and at other times.

Destination: Sent to self.

Interception: Generally not intercepted. Subclasses should use this only to see when the controller is being removed from its lists; it should call the superclass to do the actual removal.

12.4.3.4 Working with Controller Visibility

```
MSG_GEN_CONTROL_NOTIFY_INTERACTABLE,
MSG_GEN_CONTROL_NOTIFY_NOT_INTERACTABLE,
MSG_GEN_CONTROL_UNBUILD_NORMAL_UI_IF_POSSIBLE
```

Because controllers are generic objects which may or may not be interactable, **GenControlClass** has two messages to handle the interactable state. It has an additional message to close the normal UI tools, if possible. These three messages are detailed below.

■ MSG_GEN_CONTROL_NOTIFY_INTERACTABLE

```
void        MSG_GEN_CONTROL_NOTIFY_INTERACTABLE(
            GenControlInteractableFlags flags);
```

This message instructs the controller to take any actions necessary before being put on the screen and made interactable. It causes the controller to add itself to its GCN lists (if it wasn't already) and take any final steps to get ready to go on the screen.

Source: Generated Internally—do not generate externally.

Destination: Sent to self.

Parameters: *flags* A single record of flags indicating which portions of the controller will be made interactable.

Interception: May be intercepted to circumvent default behavior.

■ MSG_GEN_CONTROL_NOTIFY_NOT_INTERACTABLE

```
void    MSG_GEN_CONTROL_NOTIFY_NOT_INTERACTABLE(
        GenControlInteractableFlags flags);
```

This message indicates that the UI has determined that the controller can no longer be seen or interacted with by the user. The default behavior of the controller is to remove itself from its GCN lists for optimization.

Source: Generated Internally—do not generate externally.

12.5

Destination: Sent to self.

Parameters: *flags* A single record of flags indicating which portions of the controller are no longer interactable.

Interception: May be intercepted to circumvent default behavior.

■ MSG_GEN_CONTROL_UNBUILD_NORMAL_UI_IF_POSSIBLE

```
void    MSG_GEN_CONTROL_UNBUILD_NORMAL_UI_IF_POSSIBLE();
```

This internal message requests that the normal default controller UI gadgetry be removed and destroyed if they are not in use. This is an optimization intended to save swap space and perhaps memory space; it causes the objects to be loaded back in again when needed, however. Thus, the optimization is only suited to machines with limited swap space (it originated due to these conditions on palmtop machines).

Source: Generated internally. Do not generate externally.

Destination: Sent to self.

Interception: May be intercepted to avoid the optimization. May also need to be intercepted if a subclass creates its own UI gadgetry by intercepting MSG_GEN_CONTROL_GENERATE_UI.

12.5 GenToolControlClass

GenToolControlClass is a subclass of **GenControlClass**; the tool control allows the user to manipulate and manage all the other controllers in an application and save the configuration he or she sets up.

Advanced Topic

Previous sections of this chapter explain how to include a GenToolControl object in your application; this section details the specific structures, messages, and internals of **GenToolControlClass**. Most application programmers will never need to know this information, though some library programmers will want to read the next section.

GenToolControlClass has two instance fields beyond those inherited from **GenControlClass**. One field is the chunk handle of the Tool Location Table, and the other is the chunk handle of the Tool Group List, both defined below:

```
@instance ChunkHandle   GTCI_toolboxList;
@instance ChunkHandle   GTCI_toolGroupList;
    @default GI_states = @default | GS_ENABLED;
```

The Tool Location Table must be set up for any application that uses a GenToolControl. The structure of this table is given in “Using Tools” on page 801. The total number of allowable entries in the Tool Location Table is MAX_NUM_TOOLBOXES, which is 25. Because the GenToolControl uses one of these entries, you can have at most 24 other locations for a controller’s tools to appear.

The Tool Group List is a list of all the GenToolGroup objects associated with controllers in the application. Each controller may have exactly one GenToolGroup object for managing its tools, and all the ToolGroups must appear in this list if the GenToolControl is to manage them. The Tool Group List is described in “Using Tools” on page 801.

GenToolControlClass also uses a vardata field to store temporary information about a particular controller; the tool control uses this information when presenting the user with options of where the controller’s features may be placed. This vardata field is shown below (both the definition of the field and the structure it uses):

```
@vardata TempGenToolControlInstance
                  TEMP_GEN_TOOL_CONTROL_INSTANCE;
typedef struct {
    optr    TGTCI_curController;
    word    TGTCI_features;
    word    TGTCI_required;
    word    TGTCI_allowed;
} TempGenToolControlInstance;
```

The individual fields of the temporary structure are described below.

TGTCI_curController
The optr of the current controller whose tool options are being presented to the user. The GenToolControl allows the user to select and edit the features and location of this controller. If the user selects a new controller for editing, this field will be changed to the new controller's optr.

TGTCI_features
The mask of the currently active features of the controller.

TGTCI_required
The mask of the features which the controller object has defined as required—these must always be active and can not be "hidden" by the user. This mask is set by the controller with ATTR_GEN_CONTROL_REQUIRE_TOOLBOX_UI.

TGTCI_allowed
The mask of features that are allowed by both the controller and the application. Features set in this mask but not in *TGTCI_features* will be implemented but will be "hidden" from the user. Features not in either list will not be implemented by the controller for the application.

The GenToolControl uses a number of internal messages that should not be intercepted. You can, however, use the data structures of **GenToolControlClass** in a subclass and add functionality to the tool controller.

12.6 GenToolGroupClass

The GenToolGroup object is used to group tools for an individual controller. It is highly unlikely you will ever need to subclass or interact directly with **GenToolGroupClass**, though you will often use GenToolControl objects.

GenToolGroupClass is subclassed from **GenInteractionClass** because its primary functions are grouping and geometry management. Each GenControl object in your application should have exactly one GenToolGroup associated with it, and all the GenToolGroups should appear in the Tool Group List of the GenToolControl.

GenToolGroupClass has one instance data field in addition to those inherited from **GenInteractionClass**:

```
@instance optr    GTGI_controller;
    @default GI_states = (@default & ~GS_ENABLED);
```

This field contains the optr of the GenControl object whose tools are managed by this GenToolGroup. This field is nearly always set in your **.goc** file.

12.7

The GenToolGroup also has a temporary vardata field that indicates the color in which the tool group should be highlighted; this allows the user to better see which set of tools is being "discussed" (e.g. which set is currently being manipulated by the GenToolControl).

```
@vardata Color TEMP_TOOL_GROUP_HIGHLIGHT;
```

To set the highlight type, **GenToolGroupClass** has a single message. MSG_GEN_TOOL_GROUP_SET_HIGHLIGHT sets the color of the group's highlight; if the group should not be highlighted, the color should be set to -1.

■ MSG_GEN_TOOL_GROUP_SET_HIGHLIGHT

```
void    MSG_GEN_TOOL_GROUP_SET_HIGHLIGHT(
        ToolGroupHighlightType hlType);
```

This message sets the highlight color of the tool group. This is used almost exclusively by the GenToolControl.

Source: GenToolControl object; can also be sent by others.

Destination: The GenToolGroup whose highlight color is to be set.

Parameters: *hlType* The new highlight type to be used by the tool group. The values are TGHT_INACTIVE_HIGHLIGHT, TGHT_ACTIVE_HIGHLIGHT, and TGHT_NO_HIGHLIGHT.

Return: Nothing.

Interception: Should not be intercepted.

12.7 Other Controllers

Most system objects provide their own controllers; those controllers are documented along with the objects they control. Some controllers, however, may be used by more than one system object; these are documented here. The three classes below are **ColorSelectorClass**, **GenPageControlClass**, and **StyleSheetControlClass**.

12.7.1 ColorSelectorClass

The ColorSelector controller provides all the UI necessary to allow the user to select a color, a draw mask, and a draw pattern. The color may be selected either as an index or an RGB value. The ColorSelector must be put on the GenApplication's GAGCNLT_SELF_LOAD_OPTIONS GCN list.

The ColorSelector has the following features:

```
typedef WordFlags CSFeatures;
#define CSF_FILLED_LIST 0x10
#define CSF_INDEX       0x08
#define CSF_RGB         0x04
#define CSF_DRAW_MASK   0x02
#define CSF_PATTERN     0x01
#define CS_DEFAULT_FEATURES   (CSF_FILLED_LIST |
            CSF_INDEX | CSF_RGB | CSF_DRAW_MASK |
            CSF_PATTERN)
```

In addition, the ColorSelector has several instance fields, all shown below:

```
@instance ColorQuad     CSI_color = {0, 0, 0, 0};
            /* currently selected color */
@instance byte          CSI_colorIndeterminate;
            /* true if color is indeterminate */
@instance SystemDrawMask CSI_drawMask = SDM_0;
            /* draw mask in use by the color */
@instance byte          CSI_drawMaskIndeterminate;
            /* true if mask is indeterminate */
@instance GraphicPattern CSI_pattern = {0, 0};
            /* pattern in use by the color */
@instance byte          CSI_patternIndeterminate;
            /* true if pattern indeterminate */
```

```
@instance ColorModifiedStates CSI_states = 0;
            /* indicates which aspects have changed:
             * CMS_COLOR_CHANGED
             * CMS_DRAW_MASK_CHANGED
             * CMS_PATTERN_CHANGED */
@instance ColorToolboxPreferences CSI_toolboxPrefs
                                    = CTP_IS_POPUP;
            /* preferences for color selector:
             * CTP_INDEX_ORIENTATION
             * CTP_DRAW_MASK_ORIENTATION
             * CTP_PATTERN_ORIENTATION
             * CTP_IS_POPUP */
@vardata optr ATTR_COLOR_SELECTOR_DISABLE_OBJECT;
            /* when color selector is disabled, the
             * object named will also be disabled */
```

12.7.1.1 Messages Sent Out by the ColorSelector

```
MSG_META_COLORED_OBJECT_SET_COLOR,
MSG_META_COLORED_OBJECT_SET_DRAW_MASK,
MSG_META_COLORED_OBJECT_SET_PATTERN
```

The ColorSelector sends out three messages to its data objects. These messages each serve to set a particular attribute of the object it currently controls (the target object). The target object must be able to handle these messages if it is to use the ColorSelector.

■ MSG_META_COLORED_OBJECT_SET_COLOR

```
void    MSG_META_COLORED_OBJECT_SET_COLOR(
        ColorQuad color);
```

This message notifies the controlled object that it should set its color to the passed value.

Source: ColorSelector object.

Destination: The current Target object.

Parameters: *color* The **ColorQuad** structure describing the color to be set.

Return: Nothing.

Interception: Must be intercepted by the controlled object if it is to work with the ColorSelector controller.

■ MSG_META_COLORED_OBJECT_SET_DRAW_MASK

```
void    MSG_META_COLORED_OBJECT_SET_DRAW_MASK(
        SystemDrawMask mask);
```

This message notifies the controlled object that it should set its draw mask to the passed value.

Source: ColorSelector object.

Destination: The current Target object.

Parameters: *mask* The **SystemDrawMask** to be set as the object's draw mask.

Return: Nothing.

Interception: Must be intercepted by the controlled object if it is to work with the ColorSelector controller.

■ MSG_META_COLORED_OBJECT_SET_PATTERN

```
void    MSG_META_COLORED_OBJECT_SET_PATTERN(
        GraphicPattern pattern);
```

This message notifies the controlled object that it should sets its draw pattern to that passed.

Source: ColorSelector object.

Destination: The current Target object.

Parameters: *pattern* The **GraphicPattern** to be set as the object's draw pattern.

Return: Nothing.

Interception: Must be intercepted by the controlled object if it is to work with the ColorSelector controller.

12.7.1.2 Messages Handled by ColorSelectorClass

```
MSG_COLOR_SELECTOR_GET_COLOR,
MSG_COLOR_SELECTOR_SET_COLOR,
MSG_COLOR_SELECTOR_GET_DRAW_MASK,
MSG_COLOR_SELECTOR_SET_DRAW_MASK,
```

◆Objects

```
MSG_COLOR_SELECTOR_GET_PATTERN,
MSG_COLOR_SELECTOR_SET_PATTERN
```

The ColorSelector handles a number of messages including the normal **GenControlClass** messages it inherits. These messages allow the setting or retrieval of the instance data of the controller.

■ MSG_COLOR_SELECTOR_GET_COLOR

```
void      MSG_COLOR_SELECTOR_GET_COLOR(
          ColorQuad          *retVal);
```

This message retrieves the controller's *CSI_color* field.

Source: Unrestricted.

Destination: Any ColorSelector object.

Parameters: *retVal* A pointer to an empty ColorQuad structure.

Return: The **ColorQuad** structure pointed to by *retVal* will be the color set in *CSI_color*.

Interception: Generally not intercepted.

■ MSG_COLOR_SELECTOR_SET_COLOR

```
void      MSG_COLOR_SELECTOR_SET_COLOR(
          ColorQuad color);
```

This message sets the color in *CSI_color*.

Source: Unrestricted.

Destination: Any ColorSelector object.

Parameters: *color* The **ColorQuad** structure representing the new color for the *CSI_color* field.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_COLOR_SELECTOR_GET_DRAW_MASK

```
SystemDrawMask MSG_COLOR_SELECTOR_GET_DRAW_MASK();
```

This message retrieves the draw mask set in *CSI_drawMask*.

Source: Unrestricted.

Destination: Any ColorSelector object.

Parameters: None.

Return: The **SystemDrawMask** set in *CSI_drawMask*.

Interception: Generally not intercepted.

■ MSG_COLOR_SELECTOR_SET_DRAW_MASK

```
void        MSG_COLOR_SELECTOR_SET_DRAW_MASK(
            SystemDrawMask mask);
```

12.7 This message sets the draw mask stored in *CSI_drawMask*.

Source: Unrestricted.

Destination: Any ColorSelector object.

Parameters: *mask* The new **SystemDrawMask** to be set into *CSI_drawMask*.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_COLOR_SELECTOR_GET_PATTERN

```
GraphicPattern MSG_COLOR_SELECTOR_GET_PATTERN();
```

This message returns the pattern set in *CSI_pattern*.

Source: Unrestricted.

Destination: Any ColorSelector object.

Parameters: None.

Return: The GraphicPattern currently set in *CSI_pattern*.

Interception: Generally not intercepted.

■ MSG_COLOR_SELECTOR_SET_PATTERN

```
void        MSG_COLOR_SELECTOR_SET_PATTERN(
            GraphicPattern pattern);
```

This message sets the drawing pattern stored in *CSI_pattern*.

Source: Unrestricted.

Destination: Any ColorSelector object.

Parameters: *pattern* The **GraphicPattern** value to be set.

Return: Nothing.

Interception: Generally not intercepted.

◆ Objects

■ MSG_COLOR_SELECTOR_UPDATE_COLOR

`See colorC.goh`

Update the current color in the toolbox and normal UI. This message is normally sent from within a MSG_GEN_CONTROL_UPDATE_UI handler.

Source: Unrestricted—sent in a MSG_GEN_CONTROL_UPDATE_UI handler.

Destination: Any ColorSelector object.

Parameters: A ColorQuad structure and an indeterminate flag.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_COLOR_SELECTOR_APPLY_COLOR

`See colorC.goh`

Makes the color selector send MSG_META_COLORED_OBJECT_SET_COLOR.

Source: Unrestricted.

Destination: Any ColorSelector object.

Parameters: A Color enumeration value.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_COLOR_SELECTOR_UPDATE_FILLED_STATUS

`See colorC.goh`

Makes the color selector update the draw mask and associated UI.

Source: Unrestricted.

Destination: Any ColorSelector object.

Parameters: A **SystemDrawMask**, an indeterminate flag, and a tools update flag.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_COLOR_SELECTOR_GET_FILLED_MONIKER

`See colorC.goh`

Returns the VisMoniker that should be used to represent the “do draw” item in the color selector.

Source: Unrestricted.

Destination: Any ColorSelector object.

Parameters: None.

Return: The optr of the VisMoniker chunk or NullOptr to use the default.

Interception: Generally not intercepted.

■ MSG_COLOR_SELECTOR_GET_UNFILLED_MONIKER

`See colorC.goh`

Returns the VisMoniker that should be used to represent the "don't draw" item in the color selector.

Source: Unrestricted.

Destination: Any ColorSelector object.

Parameters: None.

Return: The optr of the VisMoniker chunk or NullOptr to use the default.

Interception: Generally not intercepted.

■ MSG_COLOR_SELECTOR_UPDATE_DRAW_MASK

`See colorC.goh`

Update the current draw mask in the toolbox and the UI. This is often sent from within a MSG_GEN_CONTROL_UPDATE_UI handler.

Source: Unrestricted—sent in a MSG_GEN_CONTROL_UPDATE_UI handler.

Destination: Any ColorSelector object.

Parameters: A **SystemDrawMask** value and an indeterminate flag.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_COLOR_SELECTOR_APPLY_DRAW_MASK

`See colorC.goh`

Generates MSG_META_COLORED_OBJECT_SET_DRAW_MASK.

Source: Unrestricted.

Destination: Any ColorSelector object.

Parameters: A **SystemDrawMask** value.

◆ Objects

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_COLOR_SELECTOR_UPDATE_PATTERN

See colorC.goh

Update the current pattern in the toolbox and the UI. This is often sent from within a MSG_GEN_CONTROL_UPDATE_UI handler.

Source: Unrestricted—sent in a MSG_GEN_CONTROL_UPDATE_UI handler.

Destination: Any ColorSelector object.

Parameters: A **GraphicPattern** value and an indeterminate flag.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_COLOR_SELECTOR_APPLY_PATTERN

See colorC.goh

Generates MSG_META_COLORED_OBJECT_SET_PATTERN.

Source: Unrestricted.

Destination: Any ColorSelector object.

Parameters: A **GraphicPattern** value.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_CS_SET_FILLED_STATUS

See colorC.goh

Tells the controller to disable itself and set the mask to zero or to re-enable itself and set the mask to 100, based on the passed flag.

Source: Unrestricted.

Destination: Any ColorSelector object.

Parameters: A **SysDrawMask** value.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_CS_SET_CF_INDEX

```
See colorC.goh
```

Sent by the color list to set a color via an index.

Source: The color list.

Destination: Any ColorSelector object.

Parameters: See **colorC.goh**.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_CS_SET_CF_RGB_RED

```
See colorC.goh
```

Sent by the color list to set a color’s red value.

Source: The color list.

Destination: Any ColorSelector object.

Parameters: see **colorC.goh**.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_CS_SET_CF_RGB_GREEN

```
See colorC.goh
```

Sent by the color list to set a color’s green value.

Source: The color list.

Destination: Any ColorSelector object.

Parameters: see **colorC.goh**.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_CS_SET_CF_RGB_BLUE

```
See colorC.goh
```

Sent by the color list to set a color’s blue value.

Source: The color list.

Destination: Any ColorSelector object.

Parameters: see **colorC.goh**.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_CS_SET_DRAW_MASK

```
See colorC.goh
```

Sets the draw mask for the color selector.

■ MSG_CS_SET_PATTERN

```
See colorC.goh
```

Sets the pattern for the color selector.

12.7.2 GenPageControlClass

GenPageControlClass provides a controller object that allows the user to go to any page in a page range. This controller provides UI gadgetry for going to a specified page, going to the next page, or going to the previous page.

The GenPageControl must be put on the GAGCNLT_SELF_LOAD_OPTIONS GCN list in the GenApplication object. It responds to the notification type MANUFACTURER_ID_GEOWORKS.GWNT_PAGE_STATE_CHANGE (sent with a **NotifyPageStateChange** structure) and sends out three messages exported from **MetaClass**.

GenPageControlClass is subclassed from GenControlClass and has the features and tools shown below. It has none of its own instance data fields.

```
typedef WordFlags GPCFeatures;
#define GPCF_GOTO_PAGE        0x0004
#define GPCF_NEXT_PAGE        0x0002
#define GPCF_PREVIOUS_PAGE    0x0001

typedef WordFlags GPCToolboxFeatures;
#define GPCTF_PREVIOUS_PAGE   0x0004
#define GPCTF_GOTO_PAGE       0x0002
#define GPCTF_NEXT_PAGE       0x0001
```

```
#define GPC_DEFAULT_FEATURES (GPCF_GOTO_PAGE |
            GPCF_NEXT_PAGE | GPCF_PREVIOUS_PAGE)
#define GPC_DEFAULT_TOOLBOX_FEATURES
            (GPCTF_GOTO_PAGE | GPCTF_NEXT_PAGE |
             GPCTF_PREVIOUS_PAGE)
@default GCI_output = (TO_APP_TARGET);
```

The messages this controller sends out must be handled by any object controlled by it.

■ MSG_META_PAGED_OBJECT_GOTO_PAGE

```
void    MSG_META_PAGED_OBJECT_GOTO_PAGE(
        word   page);
```

This message causes the paged object to go to the specified page.

Source: The GenPageControl object.

Destination: The target paged object.

Parameters: *page* The page number to which the recipient should go.

Return: Nothing.

Interception: This message must be intercepted for the paged object to interact properly with the GenPageControl object.

■ MSG_META_PAGED_OBJECT_NEXT_PAGE

```
void    MSG_META_PAGED_OBJECT_NEXT_PAGE();
```

This message indicates that the recipient should go to the next page.

Source: The GenPageControl object.

Destination: The target paged object.

Interception: This message must be intercepted for the paged object to interact properly with the GenPageControl object.

■ MSG_META_PAGED_OBJECT_PREVIOUS_PAGE

```
void    MSG_META_PAGED_OBJECT_PREVIOUS_PAGE();
```

This message indicates that the recipient should go to the previous page.

Source: The GenPageControl object.

Destination: The target paged object.

◆Objects

Interception: This message must be intercepted for the paged object to interact properly with the GenPageControl object.

■ MSG_PC_GOTO_PAGE

```
see gPageCC.goh
```

Causes the page control object to go to the specified page.

Source: Unrestricted.

Destination: The page control object.

Parameters: The page to go to.

Return: Nothing.

Interception: Should not be intercepted.

■ MSG_PC_NEXT_PAGE

```
void      MSG_PC_NEXT_PAGE();
```

Causes the page controller to go to the next page.

Source: Unrestricted.

Destination: The page control object.

Interception: Should not be intercepted.

■ MSG_PC_PREVIOUS_PAGE

```
void      MSG_PC_PREVIOUS_PAGE();
```

Causes the page controller to go to the previous page.

Source: Unrestricted.

Destination: The page control object.

Interception: Should not be intercepted.

12.7.3 The Float Format Controller

Including the Float Format controller in your application allows the user to format FP numbers into any of the system-defined formats. The controller also allows the user to define his or her own formats.

The following messages, routines and data structures all manage arrays of system-defined and user-defined formats. If you include a Float Format controller, you will need to intercept these messages and call many of these routines.

12.7.3.1 Retrieving Parameters of the Current Entry

FloatFormatGetFormatParamsWithListEntry(), FloatFormatGetFormatParamsWithToken(), FloatFormatGetFormatTokenWithName()

The Float Format controller routines use a **FormatInfoStruc** structure to hold information about a particular format. This structure is convenient to pass around to other float format routines. The Float Format controller contains one instance field, *formatInfoStrucHan*, that stores the handle to the current **FormatInfoStruc**, if one is being used by the controller.

Code Display 12-10 FormatInfoStruc

```
typedef struct {

        /*
         * FIS_signature is used internally for error-checking.
         */
        word            FIS_signature;

        /*
         * These two entries store the user defined format array for the
         * controller to work on. This array is created by FloatFormatInit().
         */
        FileHandle      FIS_userDefFmtArrayFileHan;
        VMBlockHandle   FIS_userDefFmtArrayBlkHan;

        /*
         * These two entries store the object block and format list chunk that the
         * controller resides in.
         */
        word            FIS_childBlk;
        word            FIS_chooseFmtListChunk;

        /*
         * FIS_features stores the features list of the controller.
         */
        word            FIS_features;
        /*
         * FIS_editFlag is -1 if we are editing a current user-defined entry and 0
         * if we are creating a new user-defined entry.
         */
        byte            FIS_editFlag;

        /* FIS_curSelection stores the current selection in the format list. */
        word            FIS_curSelection;

        /* FIS_curToken stores the token of the selected item in the list. */
        word            FIS_curToken;

        /* FIS_curParams stores the FormatParams of the selected item. */
        FormatParams    FIS_curParams;
} FormatInfoStruc;
```

FloatFormatGetFormatParamsWithListEntry() fills in a **FormatInfoStruc** corresponding to the passed list entry position in the Float Format controller's dynamic list. The routine must be passed a **FormatInfoStruc** with the entry position in *FIS_curSelection* already filled in. You must also have the *FIS_userDefFmtArrayFileHan* and *FIS_userDefFmtArrayBlkHan* filled in properly before calling this routine.

This routine is called whenever the user clicks on a new item, or whenever the dynamic list needs to retrieve new monikers for the list. You will also probably need to use this routine in your application code when intercepting many of the Float Format controller messages.

FloatFormatGetFormatParamsWithToken() fills in the **FormatParams** of a particular format entry when passed a format's token. The routine must be passed the **FormatInfoStruc** with *FIS_curToken* already filled in. You must also have the *FIS_userDefFmtArrayFileHan* and *FIS_userDefFmtArrayBlkHan* filled in properly before calling this routine.

FloatFormatGetFormatTokenWithName() returns the format token of a particular format when passed its name (in a **FormatInfoStruc**).

12.7.3.2 Initializing the UI

MSG_FLOAT_CTRL_UPDATE_UI, MSG_FLOAT_CTRL_REQUEST_MONIKER, FloatFormatInit(), FloatFormatInitFormatList()

FloatFormatInit() initializes a format array to serve as the storage space for user-defined formats. It must be passed the VM file handle to create the array within.

MSG_FLOAT_CTRL_UPDATE_UI is sent to the Float Format controller whenever the controller needs to perform a visual update. Your Float Format controller should intercept this message and call **FloatFormatInitFormatList()**.

FloatFormatInitFormatList() initializes the Float Format dynamic list whenever the list needs to display a new moniker. It extracts the moniker text of the selected format from the **FormatInfoStruc** structure. Make sure that this structure has the proper VM file and block handles filled in prior to calling this routine.

MSG_FLOAT_CTRL_REQUEST_MONIKER is sent to the Float Format controller whenever it needs to display the text within the controller's dynamic list. To extract the current format's moniker, intercept this message and call **FloatFormatGetFormatParamsWithListEntry()**. You can then pass the format's textual name (from *FIS_curParams.FP_formatName*) to MSG_GEN_DYNAMIC_LIST_REPLACE_ITEM_TEXT.

12.7.3.3 Setting Up Selected Formats

MSG_FLOAT_CTRL_FORMAT_SELECTED, FloatFormatProcessFormatSelected()

MSG_FLOAT_CTRL_FORMAT_SELECTED is sent to the Float Format controller whenever the user makes a new selection. This message allows your application to set up a new **FormatInfoStruc** based on the new selection. Your handler for this message needs to call **FloatFormatProcessFormatSelected()**.

12.7.3.4 User Defined Format Creation

```
MSG_FLOAT_CTRL_USER_DEF_INVOKE,
FloatFormatInvokeUserDefDB(), MSG_FLOAT_CTRL_USER_DEF_OK,
FloatFormatUserDefOK()
```

MSG_FLOAT_CTRL_USER_DEF_INVOKE is sent to the Float Format controller whenever the user has defined a new format and wishes to add it to the controller's dynamic list. Your handler for this message should call **FloatFormatInvokeUserDefDB()** in turn. If the user attempts to create a new format when MAX_FORMATS already exist, the routine will invoke an error box.

MSG_FLOAT_CTRL_USER_DEF_OK is sent to the Float Format controller to check whether the user-defined format is legal or previously duplicated. Your handler for this message should call **FloatFormatUserDefOK()** to perform this check and apply the user-defined format to the list of format entries if it is successful.

12.7.3.5 Deleting Formats

```
MSG_FLOAT_CTRL_FORMAT_DELETE, FloatFormatDelete()
```

MSG_FLOAT_CTRL_FORMAT_DELETE is sent to the Float Format controller when the user attempts to delete a user-defined entry. Your handler for this message needs to call **FloatFormatDelete()**.

12.7.3.6 Applying Formats

```
MSG_FLOAT_CTRL_FORMAT_APPLY
```

MSG_FLOAT_CTRL_FORMAT_APPLY is sent to the FloatFormat controller when the user attempts to apply a format selected in the controller to the appropriate text selection. Your handler will need to extract information from the **FormatInfoStruc** and call the appropriate text formatting routine (e.g. **FloatFloatToAsciI()**).

12.7

◆Objects

GenDocument

13

The GEOS document control objects let the programmer ignore most of the details of opening, closing, and saving files. The programmer just specifies the characteristics the application will expect of its documents. The document control presents all dialog boxes and notices to the user, and it maintains the "Save," "Save As," "Open," "New," and "Revert" triggers in the File menu.

There are three parts to the document control: the GenDocumentControl object, which maintains the user interface; the GenDocuments, each of which manages a single document; and the GenDocumentGroup object, which creates and manages GenDocument objects as needed. These classes can all be subclassed to add functionality.

You should be familiar with user interface objects in general before reading this chapter (see "The GEOS User Interface," Chapter 10 of the Concepts Book). You should also have some knowledge of the GEOS file system (see "File System," Chapter 17 of the Concepts Book) and of VM files (see "Virtual Memory," Chapter 18 of the Concepts Book).

13.1 Document Control Overview

A program that creates or uses files—in short, almost all applications—has to deal with many tasks. It has to present dialog boxes to the user so he can choose which file to open; it must give alert messages when the user tries to do something dangerous, like quit without saving; and it must notice when GEOS is being shut down and make sure it saves appropriate data.

The document control objects make these tasks much easier. They take care of some of these tasks by themselves; for example, they maintain the File menu commands and display dialog and alert boxes as needed. The Document Control objects make other application jobs much easier by sending messages at appropriate times. They can also manage several documents at once, making it much easier for applications to manage multiple documents.

Many different document control sample applications are provided with the SDK. In most cases, you should be able to get your document control by copying over code from the appropriate sample application, then customizing it.

13.1.1 The Document Control Objects

13.1

There are three different classes of objects which together constitute the document control. These are **GenDocumentControlClass**, **GenDocumentGroupClass**, and **GenDocumentClass**. A document will need at least one of each to use the document control technology. The relationship between the objects is diagramed in Figure 13-1.

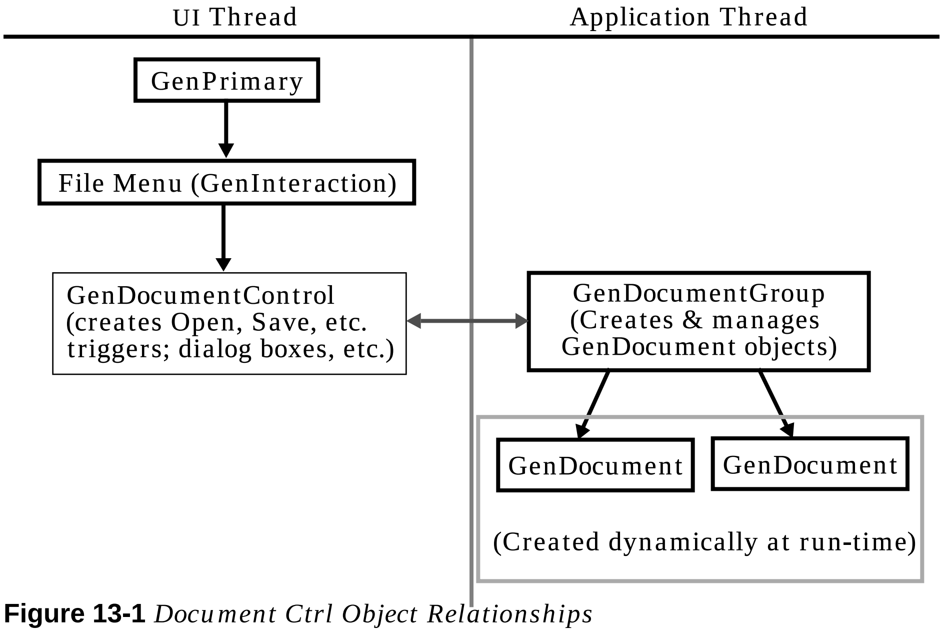

UI Thread
Application Thread
GenPrimary
File Menu (GenInteraction)
GenDocumentControl (creates Open, Save, etc. triggers; dialog boxes, etc.)
GenDocumentGroup (Creates & manages GenDocument objects)
GenDocument
GenDocument
(Created dynamically at run-time)

Figure 13-1 *Document Ctrl Object Relationships*

This diagram shows the arrangement between the different document control objects. Note that the application will not declare GenDocument objects; the GenDocumentGroup object will create them at run-time.

13.1.1.1 GenDocumentControlClass

An application will have one object of class **GenDocumentControlClass**. This object manages the user interface: It presents some dialog and alert boxes, it manages the File Selector, it creates appropriate entries for the File menu, and it updates the enabled/disabled states of these items (e.g. it disables the *Save* trigger after the file has been saved).

The GenDocumentControl is generally made a child of the file menu, which is itself a child (or descendant) of the GenPrimary. The GenDocumentControl is on the GenPrimary's active list. It has no children, but it does have an optr to the GenDocumentGroup object. By convention, the GenDocumentControl object is in the same data segment as the GenPrimary object. Certain attributes of the GenDocumentControl object will determine the characteristics of the File Selector and other UI gadgets.

GenDocumentControlClass is a subclass of **GenControlClass**. This means that you can set up toolboxes to perform the "Save," "Open," etc., actions. For more details, see "Generic UI Controllers," Chapter 12.

13.1.1.2 GenDocumentGroupClass

An application will have one object of class **GenDocumentGroupClass**. This object creates and manages the document objects. Ordinarily, the document objects belong to **GenDocumentClass**. However, if the program wishes to alter the behavior of the document object, it can create a subclass of **GenDocumentClass**. In this case, the GenDocumentGroup object will contain a pointer to the class definition of the document subclass, and will create document objects of this class as needed.

The GenDocumentGroup object is a child of any object. It does not have any children when it is declared; however, it will dynamically give itself document-object children at run-time. It is in its own data segment. Certain of its attributes determine what the attributes of its document children will be.

13.1.1.3 GenDocumentClass

Each GenDocument object manages a single open file. It keeps track of the volume, path, and filename for the document, the dirty state of the document, and other document-related information. It opens and closes files and presents file-related dialog boxes (e.g., "Save changes before closing?"). Programs often define a subclass of the **GenDocumentClass** which has additional, application-related functionality. In this case, objects of the subclass are used instead of objects of **GenDocumentClass**.

GenDocumentClass is a subclass of **GenContentClass**. Therefore, document objects can receive the output of GenView objects. GenDocument objects have all the functionality of GenContent objects. GenContent objects are themselves subclassed from **GenClass**. Very few applications will use the GenContent objects directly; for that reason, they are documented in this chapter.

Document objects are not declared in the source code; they are created at run-time by the GenDocumentGroup object. The document currently active is called the *target document*.

13.1.2 Document Control Interaction

A simple user operation will usually involve all three types of objects. For example, suppose the user selects the *Open* trigger from the File menu. The following actions will be taken:

1 The *Open* trigger sends a MSG_GEN_DOCUMENT_CONTROL_INITIATE_OPEN_DOC to the GenDocumentControl object. The GenDocumentControl object responds by displaying a file selector.

2 When the user selects a file and clicks "OK," the GenDocumentControl object gets the selected file's path from the file selector object.

3 The GenDocumentControl object sends a MSG_GEN_DOCUMENT_GROUP_OPEN_DOC to the GenDocumentGroup, passing the name and path of the file to open.

4 The GenDocumentGroup creates a document object, either of class **GenDocumentClass** or of a programmer-specified class. The

GenDocumentGroup object sends a MSG_GEN_DOCUMENT_OPEN to the document object, passing the file and path name to be opened.

5 The document object opens the file specified and handles errors appropriately. It then sends messages to the application instructing it to create the UI and initialize the file.

Most of these steps are transparent to the application programmer. Typically, an application will intercept MSG_GEN_DOCUMENT_INITIALIZE_DOCUMENT_FILE; the programmer needs only to write this handler, not all the code needed for the above steps.

13.1.3 Document Control Models

GEOS allows two distinct models of document control, a *Procedural* model and an *Object* model. While the two models use the same objects, they embody different programming philosophies.

The *Procedural* model of document control is much like traditional, non-object-oriented programming. Under this model, whenever a situation arises that needs the application's attention, the document control objects will send a message to a single object (generally the Process object). This object handles all of these situations.

Under the *Object* model of document control, the application defines a subclass of **GenDocumentClass**. This new document class has methods to handle situations needing the application's attention. This model is based on the philosophy of object-oriented programming; each document object has code to handle situations arising for that document.

The main difference between the two models is where the messages are sent. Under the Procedural model, messages are sent to the Process object; under the Object model, messages are sent to the appropriate document object. Every message sent in the Procedural model corresponds to a message sent in the Object model.

The Procedural model is simpler to use; it does not require the application to subclass objects. It is thus well-suited for simple applications which will have only one file open at a time. It may also be an easier model for programmers who are new to object-oriented programming. The Object model, on the other hand, is best suited for applications which will have may documents open at once; the application can let every document object manage a single document without worrying about other open documents.

13.1.3.1 The Procedural Model

The Procedural model is well-suited for simple, single-document applications. It is easy to use.

The *Procedural* model of document control is much like traditional, procedure-oriented programming. This model is simpler to implement than the object-model. It is well suited for simple applications which have only one document open at a time.

Under the Procedural model of document control, every time a situation arises which requires the application's attention, the document control objects will send an appropriate message to the GenDocumentGroup object's output. These messages are imported from **MetaClass**, so all objects can handle them. The application will generally use global variables for run-time data storage.

For example, when a new document needs to be initialized, the document control sends a MSG_META_DOC_OUTPUT_INITIALIZE_DOCUMENT_FILE to its output object. The output object takes any appropriate steps (e.g., storing the file handle, setting up the map block, etc.).

13.1.3.2 The Object Model

The *Object* model of document control is better suited to advanced applications and applications which will have more than one document open at a time. Under this model, the application defines a subclass of **GenDocumentClass**. This new document class has handlers for situations requiring the application's attention. It also has local variables (i.e., instance data fields) which store any information the application will need about this document.

Whenever a situation arises that needs the application's attention, the relevant document object will send a message to itself. This document object will then handle the message. For example, an application might define its own document class, **MyAppDocumentClass** (a subclass of **GenDocumentClass)**. Suppose a new document has been created and needs to be initialized. First, the GenDocumentGroup object will create a new document object by instantiating an object of **MyAppDocumentClass**.

Next, the new document object will send itself a MSG_GEN_DOCUMENT_INITIALIZE_DOCUMENT_FILE. **MyAppDocumentClass** will have a handler for this message; the handler will initialize the file as well as the document object's data structures.

13.1.3.3 Messages Under the Two Models

The simplest way to show the difference between the two models is to see how a single event is handled. This section examines one specific case, in which a document needs to be initialized; other cases are handled analogously.

Suppose a situation arises needing the application's attention; for example, a document is created and needs to be initialized. First, the document object will send an appropriate message to itself. In this case, it would send itself the message MSG_GEN_DOCUMENT_INITIALIZE_DOCUMENT_FILE. If the application uses a subclass of **GenDocumentClass** and this subclass has a handler for MSG_GEN_DOCUMENT_INITIALIZE_DOCUMENT_FILE, the messager will call that method; otherwise, the messager will call the handler defined for this message by **GenDocumentClass**. The handler in **GenDocumentClass** will find out the **GenDocumentGroup** object's output optr. If this optr is non-null, the handler will send an appropriate message (in this case, MSG_META_DOC_OUTPUT_INITIALIZE_DOCUMENT_FILE) to the output object.

Warning

If an application handles a given Object model message, the corresponding Procedural model message will not be sent.

Note that, under normal circumstances, the application will handle only one of the two messages. For example, if the application writes a handler for MSG_GEN_DOCUMENT_PHYSICAL_SAVE, the handler defined by **GenDocumentClass** will not be called; as a result, MSG_META_DOC_OUTPUT_PHYSICAL_SAVE will not be sent to the **GenDocumentGroup** object's output. This is not usually a problem, since the application will generally handle one message or the other. If, for some reason, it needs to have both messages sent, the handler for MSG_GEN_DOCUMENT_SAVE should contain a **@callsuper** instruction.

13.2 Document Control Data Fields

This section describes the attributes of the two document control classes, **GenDocumentControlClass** and **GenDocumentGroupClass**, as well as of the **GenDocumentClass**. Note that GenDocument objects are created at run-time, and their attributes are initialized by the creating GenDocumentGroup object. However, their attributes can be changed with the appropriate messages.

Many of the messages to the GenDocumentControl have corresponding messages to the GenDocumentGroup. For example, to find out the GenDocumentControl's attributes, one can either send a MSG_GEN_DOCUMENT_CONTROL_GET_ATTRS to the GenDocumentControl or send a MSG_GEN_DOCUMENT_GROUP_GET_UI_ATTRS to the GenDocumentGroup. In either case, the message will return the same result. It is sometimes more convenient to send a message to the GenDocumentGroup object; for example, a document object can do this with **@call @genParent::<message>**.

13.2.1 GenDocumentControl Data

The GenDocumentControl handles interaction between the document control and the user. It maintains the File menu entries and manages the file selector. Its attributes all relate to these duties. A complete list of the attributes follows in Code Display 13-1 along with comments and the default values.

Some of these data fields can be examined but not set by the application. Data fields for purely internal use (which are neither examined, nor set, by applications) are not listed.

Code Display 13-1 GenDocumentControl Instance Data

```
/* The GDCI_documentToken field specifies the token characters and token ID of
 * files managed by this document control. All files created by the document
 * control will have these token characters, and the File Selector object will be
 * set to allow only such files to be selected. This attribute is ignored if DOS
 * files are being opened. */
    @instance GeodeToken GDCI_documentToken = {};

/* GDCI_selectorType determines which files will be displayed by the File Selector
 * generated by this object. Only one of the options may be set. By default, only
 * documents are visible. */
    @instance GenFileSelectorTypeGDCI_selectorType = GFST_DOCUMENTS;
        /* Types available:
         *      GFST_DOCUMENTS,         GFST_EXECUTABLES,
         *      GFST_NON_GEOS_FILES,    GFST_ALL_FILES */

/* GDCI_attrs specifies certain characteristics of the file to be opened. The
 * default setting is shown below. */
    @instance GenDocumentControlAttrs    GDCI_attrs =
                        ((GDCM_SHARED_SINGLE << GDCA_MODE_OFFSET) |
                         GDCA_VM_FILE |
                         GDCA_SUPPORTS_SAVE_AS_REVERT |
                         (GDCT_NEW << GDCA_CURRENT_TASK_OFFSET))
        /* Attributes available:
         * GDCA_MODE:
         *      GDCM_VIEWER,            GDCM_SHARED_SINGLE,
         *      GDCM_SHARED_MULTIPLE
         * GDCA_CURRENT_TASK:
         *      GDCT_NONE,              GDCT_NEW,
         *      GDCT_OPEN,              GDCT_USE_TEMPLATE,
         *      GDCT_SAVE_AS,           GDCT_COPY_TO,
         *      GDCT_DIALOG,            GDCT_TYPE,
         *      GDCT_PASSWORD
         * Other fields:
         *      GDCA_MULTIPLE_OPEN_FILES,       GDCA_DOS_FILE_DENY_WRITE,
         *      GDCA_VM_FILE,                   GDCA_NATIVE,
         *      GDCA_SUPPORTS_SAVE_AS_REVERT,   GDCA_DOCUMENT_EXISTS,
         *      GDCA_DO_NOT_SAVE_FILES
         */

/* GDCI_features specifies certain extra features of the document control. The
 * default setting is shown below. */
    @instance GenDocumentControlFeatures GDCI_features =
                        (GDCF_READ_ONLY_SUPPORTS_SAVE_AS_REVERT |
                         GDCF_SINGLE_FILE_CLEAN_CAN_NEW_OPEN |
                         GDCF_SUPPORTS_TEMPLATES |
                         GDCF_SUPPORTS_USER_SETTABLE_EMPTY_DOCUMENT |
                         GDCF_SUPPORTS_USER_MAKING_SHARED_DOCUMENTS |
                         GDCF_NAME_ON_PRIMARY);
        /* Flags available:
         *      GDCF_READ_ONLY_SUPPORTS_SAVE_AS_REVERT,
         *      GDCF_SINGLE_FILE_CLEAN_CAN_NEW_OPEN,
         *      GDCF_SUPPORTS_TEMPLATES,
         *      GDCF_SUPPORTS_USER_SETTABLE_EMPTY_DOCUMENT,
         *      GDCF_SUPPORTS_USER_SETTABLE_DEFAULT_DOCUMENT,
         *      GDCF_SUPPORTS_USER_MAKING_SHARED_DOCUMENTS,
         *      GDCF_NAME_ON_PRIMARY */

/* GDCI_enableDisableList specifies objects which should be enabled whenever a
 * document is opened and disabled when all documents are closed. The field is the
 * handle of a chunk containing a list of optrs to the objects to be enabled and
 * disabled. The default value is a null handle.*/
    @instance ChunkHandle       GDCI_enableDisableList;

/* The GDCI_openGroup, GDCI_importGroup, GDCI_useTemplateGroup, GDCI_saveAsGroup,
 * GDCI_exportGroup, and GDCI_userLevelGroup attributes hold optrs to groups of UI
 * objects to be added to the “Open,” “Import,” “Use Template,” “Save As,”
 * “Export,” and “User Level”, dialog boxes, respectively. The GDCI_dialogGroup
 * field holds an optr to objects to be added to the opening dialog box. The optr
 * is to the head of a tree of UI objects. The top object in the tree should be
 * set “not usable.” Default values are all null optrs. */
    @instance optr              GDCI_openGroup;
    @instance optr              GDCI_importGroup;
    @instance optr              GDCI_useTemplateGroup;
    @instance optr              GDCI_saveAsGroup;
    @instance optr              GDCI_exportGroup;
    @instance optr              GDCI_dialogGroup;
    @instance optr              GDCI_userLevelGroup;

/* If the GDCI_features field includes “displayNameOnPrimary” but no document is
 * open, the Primary’s moniker is set to the string pointed to by the
 * GDCI_noNameText attribute. The default value is a null chunk handle. */
    @instance ChunkHandle       GDCI_noNameText;

/* If the GDCA_currentTask section of the GDCI_attrs field is set to
 * GDCT_NONE on startup, then the file specified by GDCI_defaultFile will be
 * opened (and, if necessary, created). The file is specified by a chunk handle of
 * a null-terminated string; this string should specify the file’s path relative
 * to the SP_DOCUMENT standard path.The default value is a null chunk handle,
 * indicating that if the startup value of GDCA_currentTask is GDCT_NONE, no
 * documents should be opened. */
    @instance ChunkHandle       GDCI_defaultFile;
```

```
/* GDCI_templateDir is the chunk handle of a null-terminated text string which
 * specifies a directory to hold template documents. The directory is specified
 * relative to SP_TEMPLATE. If not set by you, this defaults to the SP_TEMPLATE
 * standard path. */
    @instance ChunkHandle       GDCI_templateDir;

/* GDCI_documentGroup is an optr to the GenDocumentGroup object. You must set
 * this field. */
    @instance optr      GDCI_documentGroup;

/* GDCI_targetDocName is a character array. It is set to contain the name of
 * the current target file. This field is automatically updated by the document
 * control. */
    @instance FileLongName      GDCI_targetDocName = "";

/* The Document Control automatically displays a big dialog box at startup which
 * lets the user choose to create, open, etc. a file. Each option has a button
 * (with a picture) and an explanatory text. You can override the default graphic
 * or text by setting any of the following fields:
 */

    @instance ChunkHandle       GDCI_dialogNewText;
    @instance ChunkHandle       GDCI_dialogTemplateText;
    @instance ChunkHandle       GDCI_dialogOpenDefaultText
    @instance ChunkHandle       GDCI_dialogImportText;
    @instance ChunkHandle       GDCI_dialogOpenText;
    @instance ChunkHandle       GDCI_dialogUserLevelText;

    @instance @visMoniker       GDCI_dialogNewMoniker;
    @instance @visMoniker       GDCI_dialogTemplateMoniker;
    @instance @visMoniker       GDCI_dialogOpenDefaultText
    @instance @visMoniker       GDCI_dialogImportMoniker;
    @instance @visMoniker       GDCI_dialogOpenMoniker;
    @instance @visMoniker       GDCI_dialogUserLevelMoniker;
```

13.2.1.1 The Document Token

GDCI_documentToken, MSG_GEN_DOCUMENT_CONTROL_GET_TOKEN, MSG_GEN_DOCUMENT_GROUP_GET_TOKEN

The document control's file selector will display only those files whose document token matches the GenDocumentControl object's

13.2

GDCI_documentToken attribute. All files created by the application will have the specified document tokens. There are no messages to alter the token attributes at run-time. (If the document control is used to manage DOS files, the file selector will show all non-GEOS files.)

A token is defined by a **GeodeToken** structure. The format of this structure is shown below. The first field, *GT_chars*, will vary for each document type. The second, *GT_manufID*, will be the same for the tokens of all applications and documents created by a given company.

```
typedef struct {
      char        GT_chars[TOKEN_CHARS_LENGTH];
                           /* TOKEN_CHARS_LENGTH = 4 */
      ManufacturerID GT_manufID;
                           /* word-sized integer */
} GeodeToken;
```

The message MSG_GEN_DOCUMENT_CONTROL_GET_TOKEN instructs the GenDocumentControl object to write a copy of the document token to a specified address. The message has one argument: the address of a **GeodeToken**. MSG_GEN_DOCUMENT_GROUP_GET_TOKEN functions identically, but it is sent to the GenDocumentGroup object.

You can also find out the application's token by sending GEN_DOCUMENT_CONTROL_GET_CREATOR or GEN_DOCUMENT_GROUP_GET_CREATOR to the appropriate object. The application's token will be used as the "creator token" for any documents created by the document control.

■ MSG_GEN_DOCUMENT_CONTROL_GET_TOKEN

```
void     MSG_GEN_DOCUMENT_CONTROL_GET_TOKEN(
         GeodeToken *         token); /* address to copy token to */
```

This message gets the document token values for all documents created by this document control.

Source: Unrestricted.

Destination: Any GenDocumentControl object.

Parameters: *token* A pointer to an empty **GeodeToken** structure.

Return: The document **GeodeToken** is written to the variable whose address is passed.

◆ Objects

Interception: You should not subclass this message.

■ MSG_GEN_DOCUMENT_GROUP_GET_TOKEN

```
void      MSG_GEN_DOCUMENT_GROUP_GET_TOKEN(
          GeodeToken *        token); /* address to copy token to */
```

This is the same as MSG_GEN_DOCUMENT_CONTROL_GET_TOKEN, except that it is sent to the GenDocumentGroup object instead of the GenDocumentControl object.

Source: Unrestricted.

Destination: Any GenDocumentGroup object.

Parameters: *token* A pointer to an empty **GeodeToken** structure.

Return: The document **GeodeToken** is written to the variable whose address is passed.

Interception: You should not subclass this message.

■ MSG_GEN_DOCUMENT_CONTROL_GET_CREATOR

```
void      MSG_GEN_DOCUMENT_CONTROL_GET_CREATOR(
          GeodeToken *        token); /* address to copy token to */
```

This message gets the token for the application; this token is the “creator token” for all files created by the document control.

Source: Unrestricted.

Destination: Any GenDocumentControl object.

Parameters: *token* A pointer to an empty **GeodeToken** structure.

Return: The document **GeodeToken** is written to the variable whose address is passed.

Interception: You should not subclass this message.

■ MSG_GEN_DOCUMENT_GROUP_GET_CREATOR

```
void      MSG_GEN_DOCUMENT_GROUP_GET_CREATOR(
          GeodeToken *        token); /* address to copy token to */
```

This is the same as MSG_GEN_DOCUMENT_CONTROL_GET_CREATOR, except that it is sent to the GenDocumentGroup object instead of the GenDocumentControl object.

Source: Unrestricted.

Destination: Any GenDocumentGroup object.

Parameters: *token* A pointer to an empty **GeodeToken** structure.

Return: The document **GeodeToken** is written to the variable whose address is passed.

Interception: You should not subclass this message.

13.2.1.2 The GDCI_selectorType Field

```
GDCI_selectorType
```

The *GDCI_selectorType* field determines what files will be displayed by and can be opened with the file selector. The options are stored as a byte-sized enumerated type. The options are:

GFST_DOCUMENTS
This is the default option. The file selector will display those documents with the appropriate tokens.

GFST_EXECUTABLES
The file selector will display executable files as well as appropriate document files.

GFST_NON_GEOS_FILES
The file selector will display all non-GEOS files (and only non-GEOS files).

GFST_ALL_FILES
The file selector will display all files.

13.2.1.3 The GDCI_attrs Field

```
GDCI_attrs, MSG_GEN_DOCUMENT_CONTROL_GET_ATTRS,
MSG_GEN_DOCUMENT_GROUP_GET_UI_ATTRS
```

The GenDocumentControl object has eight attribute flags stored in the word-sized bitfield *GDCI_attrs*. They may be retrieved by sending MSG_GEN_DOCUMENT_CONTROL_GET_ATTRS to the GenDocumentControl object or by sending MSG_GEN_DOCUMENT_GROUP_GET_UI_ATTRS to the GenDocumentGroup object. The attributes are set at coding time; there is no way for an application to change the attributes at run-time, although the

GenDocumentControl will change some of the attributes to reflect its current state.

The attributes are

GDCA_MULTIPLE_OPEN_FILES
Allow several documents to be open at once. If this attribute is disabled, the “New” and “Open” triggers will be disabled when a document is open (however, see also the description of the flag GDCF_SINGLE_FILE_CLEAN_CAN_NEW_OPEN on page 881). This attribute defaults to off.

GDCA_MODE
This is a two-bit field. GDCA_MODE is a mask of all the bits in this field; the offset of this field is equal to the constant GDCA_MODE_OFFSET. The field has the following possible settings:

GDCM_VIEWER
All documents are opened in read-only mode; the New, Save, Save As, and Revert triggers are permanently disabled. Other applications can open the file for read/write access.

GDCM_SHARED_SINGLE
Documents are opened for reading and writing. When a document is open, it is marked “deny-write” so other applications can open the file only for read-only access. The user can mark a document as a “public” document, in which case the default is to open a file “read-only.” The default GDCA_MODE setting is GDCM_SHARED_SINGLE.

GDCM_SHARED_MULTIPLE
This mode is designed for documents that can have multiple writers, such as multi-user databases. Documents are ordinarily opened as in GDCM_SHARED_SINGLE mode above; however, a user can designate a file as a “multi-user” file, which means that it can be opened by several applications at once for read/write access.

The default setting of the GDCA_MODE flag is GDCM_SHARED_SINGLE. If you want a different value, first clear the two-bit field, then set the new setting, like this:

```
GDCI_attrs = (@default & ~GDCA_MODE) \
      | (GDCM_VIEWER << GDCA_MODE_OFFSET);
```

Note that the GDCA_MODE attribute has a slightly different effect if the document control manages DOS files. For details, see GDCA_DOS_FILE_DENY_WRITE below.

GDCA_DOS_FILE_DENY_WRITE
This attribute does not matter for VM files. If a DOS file is opened while the GDCA_DOS_FILE_DENY_WRITE bit is set, no other application will be able to write to that file. This is true even if the DOS file was opened for read-only access; however, if the file is a multi-user document opened in “shared-multiple” mode, other applications will be able to write to it regardless of whether the GDCA_DOS_FILE_DENY_WRITE attribute is set. By default, GDCA_DOS_FILE_DENY_WRITE is off.

GDCA_VM_FILE
This attribute specifies whether the document control objects will open GEOS Virtual Memory files (if GDCA_VM_FILE is on), or DOS files (if GDCA_VM_FILE is off). The default value is on.

GDCA_NATIVE
If this bit is set and GDCA_VM_FILE is not set, documents will be stored in the format native to the file-system.

GDCA_SUPPORTS_SAVE_AS_REVERT
This attribute is ordinarily set only for GEOS files. If the attribute is on, the application will use the backup functionality of VM files to support “Save As” and “Revert” functionality. If the attribute is off, the file will be altered whenever it is updated to disk. The default value is on. DOS files do not normally support “Save As” and “Revert.” Applications can implement “Save As” and “Revert” functionality for DOS files by defining a subclass of **GenDocumentClass** with handlers for MSG_GEN_DOCUMENT_PHYSICAL_SAVE_AS and MSG_GEN_DOCUMENT_PHYSICAL_REVERT, but this is not recommended. Ordinarily, this attribute should be off for DOS files.

GDCA_DOCUMENT_EXISTS
This attribute is set and maintained at run-time by the GenDocumentControl code. The attribute is on if at least one document is open.

GDCA_CURRENT_TASK
This three-bit attribute has a dual function: It determines the application’s behavior at start-up, and it indicates what task the application is currently performing. The mask GDCA_CURRENT_TASK is

a mask of all the bits in this field; the field's offset is equal to the constant GDCA_CURRENT_TASK_OFFSET. The possible settings are as follows:

GDCT_NONE

If a default file has been specified (see section 13.2.1.7 on page 884), that file will be opened; otherwise, the application will start with no file opened.

GDCT_NEW

A new document will be created at startup. If the GDCF_DIALOG_BOX_FOR_NEW flag is set, a dialog box will be presented at startup.

GDCT_OPEN

The “Open File” dialog box will be presented at startup.

GDCT_USE_TEMPLATE

The “Use Template” dialog box will be presented at startup.

GDCT_SAVE_AS

This is not a valid initial setting. The GDCA_CURRENT_TASK field has this setting between when a user chooses the “Save As” command and when the document is saved.

GDCT_COPY_TO

This is not a valid initial setting. The GDCA_CURRENT_TASK field has this setting between when the user chooses the “Copy To” command and when the command has been fully executed.

GDCT_DIALOG

GDCT_TYPE

GDCT_PASSWORD

None of these are valid initial settings.

If a document is passed in to be opened at startup (as, for example, when a user launches an application by double-clicking a file created by the application), that file will be opened, and the initial setting of GDCA_CURRENT_TASK will be ignored. The document control automatically maintains this bitfield to correspond to whatever action the document control is currently taking. The application can find out what the document control is doing by reading the attributes and checking this field.

GDCA_DO_NOT_SAVE_FILES

If this bit is set, the application will not be able to save files. By setting this bit, you can turn your application into a fully-functioning demo.

■ MSG_GEN_DOCUMENT_CONTROL_GET_ATTRS

```
GenDocumentControlAttrs MSG_GEN_DOCUMENT_CONTROL_GET_ATTRS();
```

Use this message to find out what the GenDocumentControl object's *GDCA_attrs* flags are. The attributes cannot be changed by a message; they can only be read.

Source: Unrestricted.

Destination: Any GenDocumentControl object.

Parameters: None.

Return: Returns a word-length bitfield containing *GDCA_attrs* flag.

Interception: You should not subclass this message.

■ MSG_GEN_DOCUMENT_GROUP_GET_UI_ATTRS

```
GenDocumentControlAttrs MSG_GEN_DOCUMENT_GROUP_GET_UI_ATTRS();
```

This message is the same as the MSG_GEN_DOCUMENT_CONTROL_GET_ATTRS message (see above) except that it is sent to the GenDocumentGroup object.

Source: Unrestricted.

Destination: Any GenDocumentGroup object.

Parameters: None.

Return: A word-length record containing the *GDCA_attrs* field.

Interception: You should not subclass this message.

13.2.1.4 The GDCI_features Flags

`GDCI_features, MSG_GEN_DOCUMENT_CONTROL_GET_FEATURES, MSG_GEN_DOCUMENT_GROUP_GET_UI_FEATURES, MSG_GEN_CONTROL_CONFIGURE_FILE_SELECTOR`

The *GDCI_features* attribute specifies whether certain optional functionality of the Document Control technology is enabled. The features are determined at coding time; there is no message to change features at run-time. To retrieve the features, send MSG_GEN_DOCUMENT_CONTROL_GET_FEATURES.

GDCF_READ_ONLY_SUPPORTS_SAVE_AS_REVERT
If this feature is on, the "Save As" and "Revert" triggers are enabled when read-only documents are opened. Once the user chooses "Save As," the new file will no longer be opened as "Read Only"; the "Save" trigger will be enabled. If this feature is off, "Save As" and "Revert" triggers are disabled for read-only files. By default, this feature is on.

GDCF_SINGLE_FILE_CLEAN_CAN_NEW_OPEN
This feature is ignored if the attribute GDCA_MULTIPLE_OPEN_FILES is on. If GDCF_SINGLE_FILE_CLEAN_CAN_NEW_OPEN is on and GDCA_MULTIPLE_OPEN_FILES is off, the "New" and "Open" triggers are enabled when the document is "clean" (i.e., the document has not been marked "dirty" since it was opened/created or saved); that is, "New" and "Open" are enabled whenever "Save" is disabled. If the user chooses "New" or "Open" when the document is "clean," the target document is closed and the new document is opened or created. If both GDCA_MULTIPLE_OPEN_FILES and GDCF_SINGLE_FILE_CLEAN_CAN_NEW_OPEN are off, the "New" and "Open" triggers are disabled whenever a document is open. By default, this attribute is on.

GDCF_SUPPORTS_TEMPLATES
If this feature is enabled, the user can save files as templates. If the user chooses the command "Use Template," a copy of the template is opened as a "new" document, and the template is left unchanged. If this feature is disabled, the application cannot create templates. By default, this feature is on.

GDCF_SUPPORTS_USER_SETTABLE_EMPTY_DOCUMENT
If this feature is enabled, the user can designate a file to be the model for all new documents. When the user chooses "New," this "model" document will be copied and the copy will be opened. By default, this feature is on.

GDCF_SUPPORTS_USER_SETTABLE_DEFAULT_DOCUMENT
If this attribute is on, the user can choose a default document (one which will automatically be opened when the application is launched). (See section 13.2.1.8 on page 884.)

GDCF_SUPPORTS_USER_MAKING_SHARED_DOCUMENTS
If this attribute is on, the user can save a document as "shared," allowing several processes to access it at once. By default, the attribute is on.

GDCF_NAME_ON_PRIMARY

If this attribute is on, the name of the target document is displayed at the top of the Primary window. The GenDocumentControl object does this by changing the moniker of the GenPrimary object to the name of the target document. If there is no open document, the GenPrimary will display the string specified by the attribute *GDCI_noNameText*. By default, this attribute is on.

GDCF_DIALOG_BOX_FOR_NEW

If this attribute is on, whenever a user selects "New," he will be presented with a dialog box. By default, this dialog box contains only triggers for "OK" and "Cancel;" the application can add other UI objects to give it functionality. If this attribute is off, whenever the user selects "New," a new document will open with the name specified by the **GenDocumentGroup** attribute *GDCI_defaultName,* if it is set; the file's name and location will be prompted for the first time the user saves the file. By default, this attribute is off.

For an added degree of control, you can use MSG_GEN_DOCUMENT_CONTROL_CONFIGURE_FILE_SELECTOR to change the attributes of the document control's file selector.

■ MSG_GEN_DOCUMENT_CONTROL_GET_FEATURES

```
GenDocumentControlFeatures MSG_GEN_DOCUMENT_CONTROL_GET_FEATURES();
```

Use this message to retrieve the current *GDCI_features* flags. The flags cannot be changed at run-time.

Source: Unrestricted.

Destination: Any GenDocumentControl object.

Parameters: None.

Return: The GenDocumentControl's *GDCI_features* flags.

Interception: You should not subclass this message.

■ MSG_GEN_DOCUMENT_GROUP_GET_UI_FEATURES

```
GenDocumentControlFeatures MSG_GEN_DOCUMENT_GROUP_GET_UI_FEATURES();
```

Use this message to retrieve the current *GDCI_features* flags. The flags cannot be changed at run-time.

Source: Unrestricted.

Destination: Any GenDocumentGroup object.

Parameters: None.

Return: The GenDocumentControl's *GDCI_features* flags.

Interception: You should not subclass this message.

■ MSG_GEN_DOCUMENT_CONTROL_CONFIGURE_FILE_SELECTOR

```
void     MSG_GEN_DOCUMENT_CONTROL_CONFIGURE_FILE_SELECTOR(
         optr                fileSelector,
         word                flags); /* GenDocumentControlAttrs */
```

Configure file selector. This message can be sub-classed to modify the behavior of the file selectors that the document control uses.

13.2.1.5 The GDCI_enableDisableList Field

```
GDCI_enableDisableList
```

The GenDocumentControl can be set to enable certain UI objects when documents are open. This is done using the *GDCI_enableDisableList* attribute. This attribute is the chunk handle of a list of object-pointers. Each of the referenced objects should start as disabled. Whenever a document is opened, a MSG_GEN_SET_ENABLED is sent to each object in the list. When the last document is closed, a MSG_GEN_SET_NOT_ENABLED is sent to each object in the list.

13.2.1.6 Adding to the Dialog Boxes

```
GDCI_openGroup, GDCI_importGroup, GDCI_useTemplateGroup,
GDCI_saveAsGroup, GDCI_exportGroup, GDCI_dialogGroup,
GDCI_userLevelGroup
```

The GenDocumentControl object manages the dialog boxes for many different user actions. The programmer can specify a tree of UI objects to be included with each of these dialog boxes. For example, to add a group of objects to the "Use Template" dialog box, the programmer should put them all in a tree (perhaps by making them all children of a GenInteraction object) and store an object-pointer to the head of the tree in the *GDCI_useTemplateGroup* attribute. The top object in the tree should be set "not usable."

13.2.1.7 The GDCI_noNameText Field

```
GDCI_noNameText
```

The GenDocumentControl object can be set to display the name of the current target document in the moniker of the **GenPrimary** window. If this feature is enabled, and no document is opened, the **GenPrimary** will have its moniker change to the string specified by *GDCI_noNameText*. If the feature GDCF_DISPLAY_NAME_ON_PRIMARY is disabled, *GDCI_noNameText* is ignored.

13.2

13.2.1.8 The GDCI_defaultFile Field

```
GDCI_defaultFile
```

If the attribute GDCA_CURRENT_TASK is initially set to GDCT_NONE and a default file is specified, the default file is automatically opened at startup. If GDCA_CURRENT_TASK is not initially set to GDCT_NONE, this attribute is ignored. This field holds the chunk handle of a null-terminated string. The string specifies the file's path relative to the SP_DOCUMENT. If the feature GDCF_SUPPORTS_USER_SETTABLE_DEFAULT_DOCUMENT is enabled, this attribute can be changed by the user at run-time. If the file specified does not exist, it is created as an empty document; if the document exists but cannot be opened, no file is opened at startup. If the named document exists but is inappropriate (e.g. it was created by another application), no document is opened at startup.

13.2.1.9 The GDCI_templateDir Field

```
GDCI_templateDir
```

If templates are supported, this is the default directory for opening and saving them. This string specifies a subdirectory to the standard path SP_TEMPLATE. If not explicitly set in the source code, the template directory will default to SP_TEMPLATE.

13.2.1.10 The GDCI_documentGroup Field

```
GDCI_documentGroup
```

The GenDocumentControl and the GenDocumentGroup communicate with each other via messages. To do this, each needs the optr of the other. *GDCI_documentGroup* is the optr of the **GenDocumentGroup** object for this application. It is set in the source code and may not be changed at run time.

13.2.1.11 The GDCI_targetDocName Field

```
GDCI_targetDocName
```

This attribute contains the name of the target document. The document control automatically sets and updates this field when necessary.

13.2.2 GenDocumentGroup Data

The GenDocumentGroup object creates and manages the document objects. In the "process" model of document control, it sends messages to the process object (or some other designated object) when the application needs to take some action. (It sends these messages even when the "object" model is being followed; however, the messages are ignored.)

A list of data fields for the GenDocumentGroup object follows in Code Display 13-2. Some of the data fields can be changed at run-time, and others cannot; a discussion of the data fields follows the listing. If a data field cannot be set or read by the application, it is not discussed.

Code Display 13-2 GenDocumentGroupClass instance data

```
/* GDGI_attrs is a record that specifies
 * certain basic characteristics of the documents to be managed. The attributes
 * are set in the source code and are not changed at run-time. The default settings
 * are below. */
    @instance GenDocumentGroupAttrs GDGI_attrs = (GDGA_VM_FILE |
                                         GDGA_SUPPORTS_AUTO_SAVE |
                                         GDGA_AUTOMATIC_CHANGE_NOTIFICATION |
                                         GDGA_AUTOMATIC_DIRTY_NOTIFICATION |
                                        GDGA_APPLICATION_THREAD |
                                        GDGA_AUTOMATIC_UNDO_INTERACTION |
                                        GDGA_CONTENT_DOES_NOT_MANAGE_CHILDREN);
        /* The following flags are available:
         *      GDGA_VM_FILE,
         *      GDGA_SUPPORTS_AUTO_SAVE,
         *      GDGA_AUTOMATIC_CHANGE_NOTIFICATION,
         *      GDGA_AUTOMATIC_DIRTY_NOTIFICATION,
         *      GDGA_APPLICATION_THREAD,
         *      GDGA_VM_FILE_CONTAINS_OBJECTS,
         *      GDGA_CONTENT_DOES_NOT_MANAGE_CHILDREN,
         *      GDGA_LARGE_CONTENT,
         *      GDGA_AUTOMATIC_UNDO_INTERACTION */

/* GDGI_untitledName is the name suggested when a new document is
 * first saved. */
    @instance ChunkHandleGDGI_untitledName;

/* The GenDocumentGroup object creates a document object for each document
 * opened. The attribute GDGI_documentClass is a pointer to the class definition
 * which will be used for document objects. By default, it points to the
 * definition of GenDocumentClass, so document objects belong to GenDocumentClass.
 * If you use a subclass of GenDocumentClass, you must change this attribute to
 * point to the new class. */
    @instance ClassStruc *GDGI_documentClass =
                                        (ClassStruc *) &GenDocumentClass;

/* Ordinarily, the Document Group creates document objects by instantiating an
 * object of the class indicated by GDGI_documentClass. However, it can be
 * instructed instead to duplicate a specific document object for each new
 * document. To arrange this, set the GDGI_genDocument to point to the document
 * object to duplicate. */
    @instance optr      GDGI_genDocument;

/* If the Procedural model is used, whenever the application needs to take an
 * action, messages will be sent to the output of the GenDocumentGroup.
 * Ordinarily, the output will be the process object. If the Object model is used,
 * this attribute is generally left as a null pointer. */
    @instance optr      GDGI_output;

/* The GenDocumentGroup object communicates with the GenDocumentControl
 * object through messages. To do this, each one needs an object-pointer to the
 * other. This is set in the source code. */
    @instance optr      GDGI_documentControl;
```

◆Objects

```
/* The GenDocument (or subclass) objects can behave as Content objects. The
 * document control can automatically connect GenDocument objects to the GenView
 * if told to do so. The GDGI_genView field is an object-pointer to a GenView
 * object. */
    @instance optr      GDGI_genView;

/* In a multiple-document model, the document control can be set up to work with
 * the display control. When this functionality is enabled, the
 * GenDocumentGroup will automatically duplicate a specified block (generally
 * one containing a GenDisplay object), attach the Display object to the specified
 * GenDisplayGroup object, and set the header for the GenDisplay to the name of
 * the document. When the document is closed, the block is freed. */
    @instance optr      GDGI_genDisplay;
/* GDGI_genDisplayGroup points to GenDisplayGroup which manages the GenDisplays. */
    @instance optr      GDGI_genDisplayGroup;

/* Each GEOS document has a protocol number, which identifies the version of
 * the application that created it. The GDGI_protocolMajor and
 * GDGI_protocolMinor attributes specify the protocol number to be assigned to
 * all documents created by the document control. */
    @instance word      GDGI_protocolMajor = 1;
    @instance word      GDGI_protocolMinor = 0;
```

13.2.2.1 The GDGI_attrs Field

GDGI_attrs, MSG_GEN_DOCUMENT_GROUP_GET_ATTRS

This attribute specifies certain characteristics of the documents to be opened. These attributes are generally set in the source code and can not be changed at run-time. They are stored in a word-sized bitfield.

GDGA_VM_FILE
This attribute is on if the documents to be opened are GEOS Virtual Memory files. By default, it is on.

GDGA_SUPPORTS_AUTO_SAVE
If this attribute is on, the documents will be periodically auto-saved. It works only with VM files (unless you subclass GenDocument to handle MSG_GEN_DOCUMENT_PHYSICAL_UPDATE; see “Working with DOS files” on page 921). It works by periodically updating the file to disk. It should probably be turned off if “Save As” and “Revert” are disabled. By default, the attribute is on. The program can temporarily disable

auto-save for a document by sending the document object MSG_GEN_DOCUMENT_DISABLE_AUTO_SAVE.

GDGA_AUTOMATIC_CHANGE_NOTIFICATION
If this attribute is on, the GenDocumentGroup object will periodically check all open documents to see if they have been changed by another process. If a document has changed, the document control will send MSG_META_DOC_OUTPUT_DOCUMENT_HAS_CHANGED to the application. This attribute is useful if the application may be reading multi-user files.

GDGA_AUTOMATIC_DIRTY_NOTIFICATION
This attribute is relevant only for GEOS files. If the attribute is on, whenever a file is marked "dirty," the file system will automatically notify the document control. The document control will then take appropriate actions (enable the "Save" trigger, etc.). The document control will also present a "Save changes before closing" dialog box if the document is closed before being saved. If GDGA_AUTOMATIC_DIRTY_NOTIFICATION is off, or if the documents are DOS files, the application will have to notify the document control when the document is dirty. It does this by sending a MSG_GEN_DOCUMENT_GROUP_MARK_DIRTY to the GenDocumentGroup object (under the procedure model), or by sending a MSG_GEN_DOCUMENT_MARK_DIRTY to the document object (under the object model). By default, GDGA_AUTOMATIC_DIRTY_NOTIFICATION is on.

GDGA_APPLICATION_THREAD
If this attribute is on, the GenDocumentGroup object is run by the application thread, as are its (document-object) children. By default, it is on.

GDGA_VM_FILE_CONTAINS_OBJECTS
If the document control manages Virtual Memory Object files, this attribute should be set to on. By default, this attribute is off.

GDGA_CONTENT_DOES_NOT_MANAGE_CHILDREN
The application's main VisContent, if any, does not manage its children. By default, this attribute is on.

GDGA_LARGE_CONTENT
The application's main VisContent uses the large model. By default, this attribute is off.

◆ Objects

GDGA_AUTOMATIC_UNDO_INTERACTION
The application sends out undo set-context messages automatically as necessary.

■ MSG_GEN_DOCUMENT_GROUP_GET_ATTRS

```
GenDocumentGroupAttrs MSG_GEN_DOCUMENT_GROUP_GET_ATTRS();
```

Use this message to find out the attributes of the GenDocumentGroup object. Note that the attributes cannot be changed at run-time; they can only be examined.

Source: Unrestricted.

Destination: Any GenDocumentGroup object.

Parameters: None.

Return: Flags in *GDGI_attrs* bitfield.

Interception: You should not subclass this message.

13.2.2.2 The GDGI_untitledName Field

```
GDGI_untitledName, MSG_GEN_DOCUMENT_GROUP_GET_DEFAULT_NAME
```

The first time a new document is saved, the document control presents a “Save As” dialog box. If the *GDGI_untitledName* field is set to point to a string, that string will be suggested as the name of the document. If the attribute is not set, no name will be suggested. The current default name can be retrieved by sending MSG_GEN_DOCUMENT_GROUP_GET_DEFAULT_NAME to the GenDocumentGroup object.

■ MSG_GEN_DOCUMENT_GROUP_GET_DEFAULT_NAME

```
GenDocumentGroupAttrs MSG_GEN_DOCUMENT_GROUP_GET_DEFAULT_NAME(
        char *buffer); /* Address to write default name */
```

This message instructs the GenDocumentGroup object to copy the *GDGI_defaultName* attribute to the specified address. In addition, the message will return the *GDGI_attrs* word of the GenDocumentGroup object. If you just want the attributes, use MSG_GEN_DOCUMENT_GROUP_GET_ATTRS.

Source: Unrestricted.

Destination: Any GenDocumentGroup object.

Parameters: *buffer* A pointer to a character buffer. This buffer should be of length FILE_LONGNAME_BUFFER_SIZE or greater.

Return: The record of flags stored in *GDGI_attrs*.

**buffer* Null-terminated name string.

Interception: You should not subclass this message.

13.2

Warnings: Make sure the buffer is long enough to hold any file name. Otherwise, the method may overwrite data after the buffer. The constant FILE_LONGNAME_BUFFER_SIZE, defined in **file.def**, is equal to the maximum file name length, counting the null terminator.

13.2.2.3 The GDGI_documentClass Field

```
GDGI_documentClass
```

Each time a document is opened, the GenDocumentGroup creates a document object. Ordinarily, the document object is a member of **GenDocumentClass**. However, sometimes the programmer will want to add functionality to the document objects, doing so by defining a subclass of **GenDocumentClass**. (For example, in the object model of document control, the program implements most functionality by defining new methods for the document class.) If this is the case, the programmer will have to make sure the GenDocumentGroup object creates document objects from the new class. One can do this by setting the *GDGI_documentClass* field to point to the class structure of the new document object class. By default, this field points to **GenDocumentClass**.

13.2.2.4 The GDGI_genDocument Field

```
GDGI_genDocument
```

Ordinarily, the document group creates new document objects by instantiating objects from the class specified in *GDGI_documentClass*. However, you can instead provide a document object for the document group to duplicate. To do this, set the *GDGI_genDocument* field to the optr of the "template" document object. This object should be marked "not usable."

Objects

13.2.2.5 The GDGI_output Field

```
GDGI_output, MSG_GEN_DOCUMENT_GROUP_GET_OUTPUT,
MSG_GEN_DOCUMENT_GROUP_SET_OUTPUT
```

Every time something happens which needs to be handled by the application, the document control notifies the application in two ways: The relevant document object sends a message to itself, and the GenDocumentGroup object sends a message to its designated output object. In the Procedural model of document control, the document-object messages are ignored, and the GenDocumentGroup messages are sent to an object (usually the process object) which has handlers for the messages. In the Object model, the *GDGI_output* attribute is left as a null pointer, and **GenDocumentClass** is subclassed to handle the messages.

■ MSG_GEN_DOCUMENT_GROUP_GET_OUTPUT

```
optr    MSG_GEN_DOCUMENT_GROUP_GET_OUTPUT();
```

Under the Procedural model of document control, the GenDocumentGroup sends messages to a designated output object. To get the optr of that output object, send this message to the GenDocumentGroup object.

Source: Unrestricted.

Destination: Any GenDocumentGroup object.

Return: Returns the optr of the document group's output object.

Interception: You should not subclass this message.

■ MSG_GEN_DOCUMENT_GROUP_SET_OUTPUT

```
void    MSG_GEN_DOCUMENT_GROUP_SET_OUTPUT(
        optr   output); /* The new recipient of the GenDocumentGroup's
               output messages */
```

Under the Procedural model of document control, the GenDocumentGroup object sends messages to a designated output object. Use this message to change the recipient of the output.

Source: Unrestricted.

Destination: Any GenDocumentGroup object.

Parameters: *output* The optr of the object which will receive the output.

Interception: You should not subclass this message.

13.2.2.6 The GDGI_documentControl Field

```
GDGI_documentControl
```

The GenDocumentControl and the GenDocumentGroup communicate with each other via messages. To do this, each needs an optr to the other. *GDCI_documentControl* is an optr to the GenDocumentGroup object for this application. It is set in the source code and is not changed at run time.

13.2.2.7 Dynamically Creating Displays

```
GDGI_genDisplay, GDGI_genDisplayGroup,
MSG_GEN_DOCUMENT_GROUP_GET_DISPLAY,
MSG_GEN_DOCUMENT_GROUP_GET_DISPLAY_GROUP
```

In a multi-document application, each document will ordinarily have its own GenDisplay object and often many other UI objects as well. The document control can be instructed to dynamically create a number of objects for each new document and destroy these objects when the document is closed.

If an application is going to have the document control create and manage GenDisplay objects, it must declare a GenDisplayGroup object. The GenDisplayGroup should be declared normally; however, it should be given no children. In the source code, the GenDocumentGroup object's *GDGI_genDisplayGroup* data field should contain an optr to the GenDisplayGroup object.

The application should also declare a template resource. This resource should contain a single generic tree; the top object in this tree should be a GenDisplay object which is set "not usable." The GenDocumentGroup object's *GDGI_genDisplay* field should contain an optr to that GenDisplay.

When a new document object is created, the document control will automatically copy the resource containing the GenDisplay referenced by *GDGI_genDisplay*, make the new GenDisplay a child of the GenDisplayGroup referenced by *GDGI_genDisplayGroup*, and set the new GenDisplay as "usable." When the document object is destroyed (because the document is closed), the document control will automatically destroy that document's copy of the resource.

■ MSG_GEN_DOCUMENT_GROUP_GET_DISPLAY

```
optr      MSG_GEN_DOCUMENT_GROUP_GET_DISPLAY();
```

The *GDGI_genDisplay* field can be set to point to a GenDisplay object. If *GDGI_genDisplay* is not a null optr, then the document control will duplicate the resource containing the referenced GenDisplay whenever a new document object is created. The duplicate GenDisplay is made a child of the GenDisplayGroup object indicated by *GDGI_genDisplayGroup*. By using this message, you can get an optr to that "template" display object. Any changes made to that object will be copied whenever a new document object is created.

Source: Unrestricted.

Destination: Any GenDocumentGroup object.

Return: Returns the optr of the "template" GenDisplay.

Interception: You should not subclass this message.

■ MSG_GEN_DOCUMENT_GROUP_GET_DISPLAY_GROUP

```
optr      MSG_GEN_DOCUMENT_GROUP_GET_DISPLAY_GROUP();
```

If a GenDisplayGroup object is used to manage GenDisplay objects, the GenDocumentGroup object will contain an optr to the GenDisplayGroup. By using this message, you can get an optr to the GenDisplayGroup object.

Source: Unrestricted.

Destination: Any GenDocumentGroup object.

Return: Returns the optr of the GenDocumentGroup object.

Interception: You should not subclass this message.

13.2.2.8 Connecting Documents with a GenView

`GDGI_genView, MSG_GEN_DOCUMENT_GROUP_GET_VIEW`

The document control can be instructed to automatically connect the output of a GenView to the document object associated with the view. That way, the document object can handle all the messages relating to the view. This is naturally only done when the application is using the Object model; if it is using the Procedural model, a GenView will most likely send its messages to the Process object.

There are two ways to enable this functionality. One way is appropriate only to single-document applications; the other is appropriate to multi-document applications.

A single-document application using the Object model should declare the GenView normally as part of the generic tree. (It might well be placed on the GenDocumentControl object's *GDCI_enableDisableList*.) The source code should set the *GDGI_genView* field to be an optr to the GenView. When a document is opened, the document control will automatically set the GenView object to direct its output to the document object.

A multi-document application using the Object model should use the document control's ability to create and manage GenDisplay objects. The application will have a resource which is duplicated for each open document. This resource will contain a generic tree, at the head of which is a GenDisplay. To use a GenView, all the application has to do is put a GenView in the tree headed by that GenDisplay, and set *GDGI_genView* to point to that GenView. When a document object is created, the document control will automatically have the new GenView (in the duplicate resource) send its output to the new GenDocument.

■ MSG_GEN_DOCUMENT_GROUP_GET_VIEW

```
optr MSG_GEN_DOCUMENT_GROUP_GET_VIEW();
```

The GenDocumentGroup object can be set to automatically link document objects to **GenView** objects. Use this message to find out what the designated **GenView** is. If there is no such **GenView**, this message will return a null optr.

Source: Unrestricted.

Destination: Any GenDocumentGroup object.

Return: The optr of the **GenView** object (specified in *GDGI_genView).*

Interception: You should not subclass this message.

13.2.2.9 Document Protocols

```
GDGI_protocolMajor, GDGI_protocolMinor,
MSG_GEN_DOCUMENT_GROUP_GET_PROTOCOL
```

Every GEOS file (and each application that creates such files) has a protocol associated with it. Protocols are used to make sure an application knows if a given document was created by a different version of the application. They are stored in the file's FEA_PROTOCOL extended attributes (see page 650 of "File System," Chapter 17 of the Concepts Book). The protocol is of the form "MAJOR.MINOR," where both "MAJOR" and "MINOR" are 16-bit unsigned integers. The application's protocol is specified by the *GDGI_protocolMajor* and *GDGI_protocolMinor* attributes of the GenDocumentGroup object.

All documents created by an application will have the application's protocol number. If a document has the same major protocol number as the application but a lower minor protocol number, the document is compatible with the application. If the document has a lower major protocol number, the document is incompatible with the application; it can be opened only if a routine has been defined to upgrade the document. If the document has a higher protocol than the application (i.e. its major protocol number is higher, or it has the same major protocol number and a higher minor protocol number), the document control will not open the file; it will present an error message. By default, the GenDocumentGroup object has a *GDGI_protocolMajor* of one and a *GDGI_protocolMinor* of zero.

When the user opens an earlier but compatible document, the GenDocumentGroup opens the file and attaches it to a document object. Then, the (newly-created) document object sends itself MSG_GEN_DOCUMENT_UPDATE_EARLIER_COMPATIBLE_DOCUMENT; the GenDocumentGroup then sends its output object the message MSG_META_DOC_OUTPUT_UPDATE_EARLIER_COMPATIBLE_DOCUMENT. If neither of these messages is handled, the document will be opened as if it were of the current protocol (since it is compatible). Often an application will not handle these messages.

If the user tries to open an earlier and incompatible document, the GenDocumentGroup opens the file and attaches it to a document object. Then, the document object sends a MSG_GEN_DOCUMENT_UPDATE_EARLIER_INCOMPATIBLE_DOCUMENT to itself, and the GenDocumentGroup sends a MSG_META_DOC_OUTPUT_UPDATE_EARLIER_INCOMPATIBLE_DOCUMENT to its output. If neither message is handled, the GenDocumentGroup closes the file unchanged and removes the document object, and the GenDocumentControl presents an error message (since the document is incompatible). If either message is handled, the document will be opened normally after the handler exits.

Note that the document control will not automatically change the protocol number for a file after it has been updated. If you wish this done, you should have the handler for the message call one of the routines to change the FEA_PROTOCOL extended attribute.

■ MSG_GEN_DOCUMENT_GROUP_GET_PROTOCOL

```
dword    MSG_GEN_DOCUMENT_GROUP_GET_PROTOCOL();
```

Use this message to get the protocol number associated with the GenDocumentGroup. A protocol number is composed of two parts, a major component and a minor component. This message returns a double-word; the high word is the major component, and the low word is the minor component.

Source: Unrestricted.

Destination: GenDocumentGroupClass.

Return: Returns a dword-sized value; the high word contains the major protocol number, and the low word contains the minor protocol number.

Interception: You should not subclass this message.

13.2.3 GenDocument Attributes

There are very few **GenDocumentClass** attributes that you will need to be concerned with. The GenDocumentGroup object creates and updates document objects as needed. Ordinarily, the application will not look at the **GenDocumentClass** instance data. If the program defines a subclass of **GenDocumentClass**, the subclass's methods should use only the subclass's instance data.

There is only one attribute which the program should change at run-time, and that is the GDA_PREVENT_AUTO_SAVE bit of the *GDI_attrs* field. This bit can be changed with the messages

MSG_GEN_DOCUMENT_DISABLE_AUTO_SAVE and MSG_GEN_DOCUMENT_ENABLE_AUTO_SAVE.

GenDocumentClass is a subclass of **GenContentClass** and has all the functionality of that class. Since GenContent objects are rarely used directly, the class does not have its own chapter; instead, it is documented in section 13.2.3.4 on page 902. The main thing to know about the GenContent is that, like a VisContent, it is displayed in a GenView and can have visible children. It can also have generic children, though it may not have both visible and generic children at the same time.

13.2.3.1 The GDI_attrs Field

```
GDI_attrs, MSG_GEN_DOCUMENT_GET_ATTRS,
MSG_GEN_DOCUMENT_ENABLE_AUTO_SAVE,
MSG_GEN_DOCUMENT_DISABLE_AUTO_SAVE,
MSG_GEN_DOCUMENT_AUTO_SAVE
```

The *GDI_attrs* word contains flags indicating the status of the document. The application can read or change any of these attributes; however, only the attribute GDA_PREVENT_AUTO_SAVE should actually be changed at run-time.

GDA_READ_ONLY
This attribute is set if the document in question is opened for read-only access.

GDA_READ_WRITE
This attribute is set if the document is opened for read/write access.

GDA_FORCE_DENY_WRITE
If this attribute is set, while the document is open, no other process will be allowed to open that document for read/write access.

GDA_SHARED_MULTIPLE
The document is opened in “shared multiple” mode.

GDA_SHARED_SINGLE
The document is opened in “shared single” mode.

GDA_UNTITLED
The document is newly-created and has not yet been saved; it is still untitled.

GDA_DIRTY The document has been marked dirty since the last time it was saved.

GDA_CLOSING
The document is in the process of being closed.

GDA_ATTACH_TO_DIRTY_FILE
The document object is being attached to a dirty file (e.g., when restarting GEOS).

GDA_SAVE_FAILED
The user attempted to save the document, and it could not be saved (e.g., someone else denied write access, or the volume was no longer accessible).

GDA_OPENING
The document is in the process of being opened.

GDA_AUTO_SAVE_STOPPED
Auto-save was stopped while in progress.

GDA_PREVENT_AUTO_SAVE
This bit can be changed by the application at run-time. While the bit is on, auto-save is disabled.

■ MSG_GEN_DOCUMENT_GET_ATTRS

```
GenDocumentAttrs MSG_GEN_DOCUMENT_GET_ATTRS();
```

Use this message to get the *GDI_attrs* flags for a given document. These attribute flags give information about the document's permissions as well as about any operations currently in progress.

Source: Unrestricted—objects subclassed from **GenDocumentClass** often send this message to themselves.

Destination: Any GenDocument object.

Return: The object's word-sized *GDI_attrs* field.

Interception: You should not subclass this message.

■ MSG_GEN_DOCUMENT_DISABLE_AUTO_SAVE

```
void        MSG_GEN_DOCUMENT_DISABLE_AUTO_SAVE();
```

Sometimes an application needs to temporarily disable auto-save for a specific document (for example, if it is in the middle of making elaborate changes to the file). It can do this by sending this message to the document object. The document's GDA_PREVENT_AUTO_SAVE bit will be turned on, and auto-save will be disabled until the document receives a MSG_GEN_DOCUMENT_ENABLE_AUTO_SAVE.

13.2

Source: Unrestricted—objects subclassed from **GenDocumentClass** often send this message to themselves.

Destination: Any GenDocument object.

Interception: You should not subclass this message.

■ MSG_GEN_DOCUMENT_ENABLE_AUTO_SAVE

```
void        MSG_GEN_DOCUMENT_ENABLE_AUTO_SAVE();
```

This message turns off a document's GDA_PREVENT_AUTO_SAVE bit. If the bit is already off (i.e., auto-save is enabled), the message has no effect.

Source: Unrestricted—objects subclassed from **GenDocumentClass** often send this message to themselves.

Destination: Any GenDocument object.

Interception: You should not subclass this message.

■ MSG_GEN_DOCUMENT_AUTO_SAVE

```
void        MSG_GEN_DOCUMENT_AUTO_SAVE();
```

This message forces the document object to immediately auto-save its file.

Source: Unrestricted. The document object may send this message to itself.

Destination: Any GenDocument object.

Interception: This message is not generally subclassed.

13.2.3.2 The GDI_operation Attribute

`GDI_operation, MSG_GEN_DOCUMENT_GET_OPERATION`

A single user action can result in many routines being called and many messages being sent out. To help keep track of what's going on, **GenDocumentClass** has a byte-length field, *GDI_operation*. If the document control is in the midst of handling a user action for a given document, it will set the *GDI_operation* byte accordingly. The current operation is a member of the **GenDocumentOperation** enumerated type. This type has the following possible values:

GDO_NORMAL
This is the usual setting. If the document is not currently handling a user action, this is the setting.

GDO_SAVE_AS
When the user chooses "Save As," the byte is set to this value. It remains at this value until the new document has been opened and saved.

GDO_REVERT
The setting from the time the user chooses "Revert" until the file has been restored to its last-saved state.

GDO_REVERT_QUICK
The setting from the time the user chooses "restore from backup" until the file has been restored.

GDO_ATTACH
The setting while the UI is being created for and attached to a given document. When a file document is created or opened, the *GDI_operation* field is set to this after GEOS has opened the file, and it remains at this setting until the application finishes attaching the UI.

GDO_DETACH
The setting while the UI is being detached from a given document, but before the actual file is closed.

GDO_NEW
The setting while a new file is being created. When the file is created and initialized, the *GDI_operation* will change to GDO_ATTACH.

GDO_OPEN The setting while an existing file is being opened. When the file is created and initialized, the *GDI_operation* will change to GDO_ATTACH.

GDO_SAVE The setting while a document is being saved.

GDO_CLOSE After the UI has been detached, the *GDI_operation* byte is set to this value until the document object is destroyed.

GDO_AUTO_SAVE
The setting while a file is being updated (i.e. auto-saved) to disk.

■ MSG_GEN_DOCUMENT_GET_OPERATION

```
GenDocumentOperation MSG_GEN_DOCUMENT_GET_OPERATION();
```

Use this message to find out what user action a given document object is in the midst of processing. This is useful if you are handling some message and want to find out the context in which that message was sent. Note that although the message returns a word-length value, the *GDI_operation* enumerated type is byte-length; it is thus safe to cast the return value to a byte-length variable.

Source: Unrestricted—objects subclassed from **GenDocumentClass** often send this message to themselves.

Destination: Any GenDocument object.

Return: Returns a member of the **GenDocumentOperation** enumerated type corresponding to the document object's current operation.

Interception: You should not subclass this message.

13.2.3.3 File Information

```
GDI_fileHandle, GDI_diskHandle, GDI_volumeName,
GDI_fileName, MSG_GEN_DOCUMENT_GET_FILE_NAME,
MSG_GEN_DOCUMENT_GET_FILE_HANDLE
```

The document object stores certain data about the file associated with it. In particular, the instance data records the document's path, its full file name, and the handles of the file and the disk volume containing the file. This data can be retrieved by sending messages to the document object.

■ MSG_GEN_DOCUMENT_GET_FILE_NAME

```
void       MSG_GEN_DOCUMENT_GET_FILE_NAME(
           char *buffer); /* Address to write file name to */
```

This message instructs a GenDocument to write the name of its file (without the path) to the specified address.

Source: Unrestricted—objects subclassed from **GenDocumentClass** often send this message to themselves.

Destination: Any GenDocument object

Parameters: *buffer* Buffer of length FILE_LONGNAME_BUFFER_SIZE.

Return: Writes file's virtual name into the passed buffer as a null-terminated string.

Warnings: Make sure the buffer passed is of length FILE_LONGNAME_BUFFER_SIZE; otherwise the method might overwrite other data.

Interception: You should not subclass this message.

■ MSG_GEN_DOCUMENT_GET_FILE_HANDLE

```
FileHandle MSG_GEN_DOCUMENT_GET_FILE_HANDLE();
```

This message returns the handle of the file associated with a given GenDocument object.

Source: Unrestricted—objects subclassed from **GenDocumentClass** often send this message to themselves.

Destination: Any GenDocument object.

Return: Returns handle of file associated with that document object.

Interception: You should not subclass this message.

13.2.3.4 GenContentClass

The GenContent generic object is similar to the VisContent visible object in that it interacts directly with the GenView. While the VisContent allows an application to display a visible hierarchy of objects within the view, however, the GenContent allows either generic or visible object hierarchies or to be displayed. This is the one case where you may ordinarily have visible objects be children of a generic object. Note that you should not have both visible and generic objects as children of the same GenContent; if you do so, results are undefined.

GenContentClass is a subclass of **GenClass** and therefore inherits all the instance data, messages, and hints of all generic objects. The GenContent also has two other instance data fields, however; these are

```
@instance byte    GCI_attrs = 0;
@instance optr    GCI_genView;
```

13.2

The *GCI_attrs* field contains a record of **VisContentAttrs** and is used by document objects for visual updates and interaction with the GenView. This record may be retrieved with MSG_GEN_CONTENT_GET_ATTRS or set with MSG_GEN_CONTENT_SET_ATTRS.

The *GCI_genView* field contains the optr of the GenView object displaying the GenContent. This, too, is used by document objects to manage interaction with the GenView.

■ MSG_GEN_CONTENT_GET_ATTRS

```
byte      MSG_GEN_CONTENT_GET_ATTRS();
```

This message returns the record of **VisContentAttrs** set in the GenContent's *GCI_attrs* field.

Source: Unrestricted.

Destination: Any GenContent object

Parameters: None.

Return: The *GCI_attrs* settings.

Interception: Unlikely.

■ MSG_GEN_CONTENT_SET_ATTRS

```
void      MSG_GEN_CONTENT_SET_ATTRS(
          byte   attrsToSet,
          byte   attrsToClear);
```

This message sets the attributes in the GenContent's *GCI_attrs* record.

Source: Unrestricted.

Destination: Any GenContent object.

Parameters: *attrsToSet* A record of **VisContentAttrs** indicating which flags should be set. Those set in *attrsToSet* will be set in *GCI_attrs*.

attrsToClear A record of **VisContentAttrs** indicating which flags should be cleared. Those cleared in *attrsToClear* will be cleared in *GCI_attrs*. Note that if a flag is set in both parameters, it will end up cleared.

Return: Nothing.

Interception: Unlikely.

13.3 Basic DC Messages

The document control objects use messages for many things. Since well over half a dozen classes of objects (counting file selectors, GenDisplayGroup objects, menu triggers, etc.) and far more actual objects are involved in intricate tasks, many messages are continually sent back and forth. Most of these messages are transparent to the programmer. The programmer need only know about them if the program subclasses a message to add functionality to it; this is an advanced technique which few programs will ever need to use.

There are two basic types of messages the programmer needs to know about. First, there are messages which are sent to document control objects; these objects may query information, toggle some functionality, or otherwise instruct the DC objects to take some action. Second, there are messages the DC objects send when the programmer's code needs to take some action. Each type of message is treated in a separate section.

13.3.1 Other Document Group Messages

The following are the messages a program might ordinarily send to the GenDocumentGroup object. Many of these messages request information about the GenDocumentControl object or the target document; others request information about the GenDocumentGroup object or instruct it to take actions. Many of the messages require, as an argument, an optr to a document object; however, a null object-pointer can be passed, thus indicating the target document. This is especially useful under single-document models; the application doesn't need to keep track of the document object's optr, since it is always the target document.

■ MSG_GEN_DOCUMENT_GROUP_MARK_DIRTY

```
void        MSG_GEN_DOCUMENT_GROUP_MARK_DIRTY(
            optr   document); /* document to mark dirty */
```

This message notifies the GenDocumentGroup object that the specified document has been dirtied. The GenDocumentGroup will enable and disable file menu triggers as appropriate. If the argument is a null pointer, the target document will be marked dirty.

Source: Unrestricted.

Destination: Any GenDocumentGroup object.

Parameters: *document* optr of document to mark dirty. If a null optr is passed, the target document will be dirtied.

Interception: You should not subclass this message.

Tips: If the document is a VM file and the GenDocumentGroup attribute GDGA_AUTOMATIC_DIRTY_NOTIFICATION is set, the VM routines will notify the GenDocumentGroup that the document has been dirtied whenever the **VMDirty()** (or **DBDirty()**, **CellDirty()**, etc.) routine is called. However, if you change a data cache without changing the actual file, you should send this message (or MSG_GEN_DOCUMENT_GROUP_MARK_DIRTY_BY_FILE) to insure that the changes to the cache will be saved.

■ MSG_GEN_DOCUMENT_GROUP_MARK_DIRTY_BY_FILE

```
void        MSG_GEN_DOCUMENT_GROUP_MARK_DIRTY_BY_FILE(
            FileHandle file); /* file handle of document to mark dirty */
```

This message notifies the GenDocumentGroup that the file with the specified handle has been dirtied. The GenDocumentGroup will enable and disable file menu triggers as appropriate. The document is specified by file handle, not document optr.

Source: Unrestricted

Destination: Any GenDocumentGroup object.

Parameters: *file* The handle of the file to dirty.

Interception: You should not subclass this message.

Tips: If the document is a VM file and the GenDocumentGroup attribute GDGA_AUTOMATIC_DIRTY_NOTIFICATION is set, the VM routines will notify the GenDocumentGroup that the document has been dirtied whenever the **VMDirty()** (or **DBDirty()**, **CellDirty()**, etc.) routine is called. However, if you change a data cache without changing the actual file, you should send this message (or MSG_GEN_DOCUMENT_GROUP_MARK_DIRTY) to insure that the changes to the cache will be saved.

■ MSG_GEN_DOCUMENT_GROUP_GET_DOC_BY_FILE

```
optr    MSG_GEN_DOCUMENT_GROUP_GET_DOC_BY_FILE(
        FileHandle file);
```

Use this message if you know the file handle of a document and you need to get an object-pointer to the document object corresponding to the file. In the inverse situation (you know the object, and need to get the file handle), send MSG_GEN_DOCUMENT_GET_FILE_HANDLE directly to the document object.

Source: Unrestricted.

Destination: Any GenDocumentGroup object.

Parameters: *file* The handle of file whose document object is needed.

Return: Returns the optr of document object

Interception: You should not subclass this message.

■ MSG_GEN_DOCUMENT_GROUP_SAVE_AS_CANCELLED

```
void    MSG_GEN_DOCUMENT_GROUP_SAVE_AS_CANCELLED();
```

If you are in the midst of handling a “Save As” operation, and you need to cancel it, send this message to the GenDocumentGroup object.

Source: Unrestricted

Destination: Any GenDocumentGroup object.

Interception: This message is not generally subclassed.

◆ Objects

13.3.2 From the Doc Control Objects

Often the document control will need to notify the application to take an action. For example, when a document is created, the application needs to be told to initialize the document and the user interface. There are two basic models for handling these situations: the “Procedure” model and the “Object” model. (The differences between these models are discussed at more length in the section “Document Control Models” on page 867.) Each model has its own way of messaging.

Under the Procedure model, every time the application needs to be notified, the document control sends a message to the GenDocumentGroup’s output object, which is ordinarily the process object. Under the Object model, the affected document object will send a message to itself; this message has no handler under **GenDocumentClass**, so the application must use a subclass of **GenDocumentClass** with handlers for these messages.

A single user action can generate several messages. For example, when the user opens a document, four messages are sent: MSG_..._PHYSICAL_OPEN, MSG_..._READ_CACHED_DATA_FROM_FILE, MSG_..._CREATE_UI_FOR_DOCUMENT, and MSG_..._ATTACH_UI_TO_DOCUMENT. Furthermore, a given message might be sent as the result of several different user actions; for example, the message MSG_..._CREATE_UI_FOR_DOCUMENT is sent when the user creates a new file or opens an existing one. If a handler needs to know what user action precipitated a given message, it can send a MSG_GEN_DOCUMENT_GET_OPERATION to the document object.

13.3.2.1 Messages Handled under the Procedure Model

Whenever the application needs to be notified to take an action, a message will be sent to the GenDocumentGroup’s output object. Two arguments accompany such messages: A pointer to the relevant document object and the handle of the file associated with that document. All of these messages are exported from **MetaClass**, so they can be handled by objects of any class.

Each of these messages corresponds to a document-model message, all of which are described in section 13.3.2.2 on page 911. These are just the basic messages; for more advanced functionality, see the message listings in the advanced section.

■ MSG_META_DOC_OUTPUT_INITIALIZE_DOCUMENT_FILE

```
void      MSG_META_DOC_OUTPUT_INITIALIZE_FILE(
          optr                 document, /* document object to initialize */
          FileHandle           file); /* Handle of file to initialize */
```

The GenDocumentGroup object sends this message out when a new document has been created and needs to be initialized. Applications which use VM files will allocate the map block and initialize it. If an application maintains data caches for its files, it should initialize the caches at this point.

Note that the handler for this message should not take any UI-related actions. These should be left to the handlers for MSG_META_DOC_OUTPUT_CREATE_UI_FOR_DOCUMENT and MSG_META_DOC_OUTPUT_ATTACH_UI_FOR_DOCUMENT.

Source: The GenDocumentGroup object.

Destination: The output of GenDocumentGroup (usually the Process object).

Parameters: *document* The optr of the document object which has just been created.

file The **FileHandle** of the file which has just been created or opened.

Interception: If you are using the Procedure model, you must write a handler for this message in whatever class will be receiving it (usually the process class).

■ MSG_META_DOC_OUTPUT_CREATE_UI_FOR_DOCUMENT

```
void      MSG_META_DOC_OUTPUT_CREATE_UI_FOR_DOCUMENT(
          optr                 document, /* Pointer to document object */
          FileHandle           file); /* Handle of file associated with
                                    * document object */
```

The GenDocumentGroup object sends this message after a document has been created or opened. Before this message is sent, the GenDocumentControl object will enable those objects on its *GDCI_enableDisableList*, and the GenDocumentGroup object will copy the GenDisplay resource for the document (if one is defined).

Applications that use dynamic UI objects will commonly respond to this message by creating the objects for the newly-opened document. Applications that use static UI objects will commonly respond to this message by enabling the objects.

Source: The GenDocumentGroup object.

Destination: The output of GenDocumentGroup (usually the Process object).

Parameters: *document* The optr of the appropriate document object.

file The FileHandle of the appropriate file.

Interception: If you are using the Procedure model, you must write a handler for this message in whatever class will be receiving it (usually the process class).

■ MSG_META_DOC_OUTPUT_ATTACH_UI_TO_DOCUMENT

```
void    MSG_META_DOC_OUTPUT_ATTACH_UI_TO_DOCUMENT(
        optr                document, /* optr of document object */
        FileHandle          file); /* handle of file for this document */
```

The GenDocumentGroup object sends this message when the UI for a newly-opened document has been created. It also sends this message when re-opening a document as part of restoring GEOS from a state file. Applications may respond to this by attaching dynamic UI objects and setting the values of static UI objects.

Source: The GenDocumentGroup object.

Destination: Output of GenDocumentGroup (usually the Process object).

Parameters: *document* The optr of the document object.

file The FileHandle of the appropriate file.

■ MSG_META_DOC_OUTPUT_DETACH_UI_FROM_DOCUMENT

```
void    MSG_META_DOC_OUTPUT_DETACH_UI_FROM_DOCUMENT(
        optr                document, /* pointer to document object */
        FileHandle          file); /* handle of file for this document */
```

The GenDocumentGroup sends this message when a document is being closed, whether because a user closes the file or because the application is being closed. It also sends this message when GEOS is in the process of saving itself to a state file prior to shutting down. Applications generally respond to this by detaching dynamic UI objects. Note that the GenDocumentControl object will automatically disable any objects in its *GDCI_enableDisableList*.

Source: The GenDocumentGroup object.

Destination: Output of GenDocumentGroup (usually the Process object).

Parameters: *document* The optr of the appropriate document object.

file The FileHandle of the appropriate file.

Interception: If you are using the Procedure model, you must write a handler for this message in whatever class will be receiving it (usually the process class).

13.3

■ MSG_META_DOC_OUTPUT_DESTROY_UI_FOR_DOCUMENT

```
void        MSG_META_DOC_OUTPUT_DESTROY_UI_FOR_DOCUMENT(
            optr   document, /* pointer to document object */
            FileHandle file); /* handle of file for this document */
```

The GenDocumentGroup object sends this message out when a document is being closed, whether because a user closes the file or because the application is being closed. Applications will generally disable static display objects and delete dynamic display objects. Note that the GenDocumentControl object will automatically disable all objects in its *GDCI_enableDisableList*, and the GenDocumentGroup will delete the display block it created for a document, if any.

Source: The GenDocumentGroup object.

Destination: Output of GenDocumentGroup (usually the Process object).

Parameters: *document* The optr of the appropriate document object.

file The FileHandle of the appropriate file.

Interception: If you are using the Procedure model, you must write a handler for this message in whatever class will be receiving it (usually the process class).

■ MSG_META_DOC_OUTPUT_ATTACH_FAILED

```
void        MSG_META_DOC_OUTPUT_ATTACH_FAILED(
            optr   document, /* pointer to document object */
            FileHandle File); /* (former) handle of file for this document */
```

When GEOS restores itself from state, the document control tries to reattach all documents which were attached when GEOS was shut down. If this is impossible (as for example if a document was deleted after GEOS shut down), the GenDocumentGroup object will send this message to its output.

Source: The GenDocumentGroup object.

Destination: Output of GenDocumentGroup (usually the Process object).

Parameters: *document* The optr of the appropriate document object.

file The FileHandle of the appropriate file.

Interception: If you are using the Procedure model, you must write a handler for this message in whatever class will be receiving it (usually the process class).

13.3.2.2 Messages Handled under the Object Model

If an application uses the Object model of document control, it will generally not handle the messages to the GenDocumentGroup's output. Instead, it will define a subclass of **GenDocumentClass**; this subclass will have methods for those situations which require the application's attention. Note that **GenDocumentClass** does not have handlers for any of these messages; if the application does not define a method for a given message, that message will have no effect.

■ MSG_GEN_DOCUMENT_INITIALIZE_DOCUMENT_FILE

```
Boolean   MSG_GEN_DOCUMENT_INITIALIZE_DOCUMENT_FILE();
```

When a new document is created, the document object sends this message to itself. VM file based applications will generally respond to this message by allocating and initializing the map block. DOS file based applications will commonly initialize data structures for a default file. If an application maintains data caches for its files, it should initialize the caches at this point.

The application should not take any UI-related actions; those should be postponed until the messages MSG_GEN_DOCUMENT_CREATE_UI_FOR_DOCUMENT and MSG_GEN_DOCUMENT_ATTACH_UI_TO_DOCUMENT are received.

Source: A GenDocument object.

Destination: The document object sends this message to itself.

Parameters: Nothing.

Return: If the handler could not initialize the file, it should return *true*; the document control will then destroy the new file.

13.3

■ MSG_GEN_DOCUMENT_CREATE_UI_FOR_DOCUMENT

```
void        MSG_GEN_DOCUMENT_CREATE_UI_FOR_DOCUMENT();
```

The GenDocument object sends this message to itself after a document has been created or opened. Before this message is sent, the GenDocumentControl object will enable those objects on its *GDCI_enableDisableList*, and the GenDocumentGroup object will copy the GenDisplay block for the document (if one is defined).

Applications that use dynamic UI objects will commonly respond to this message by creating the objects for the newly-opened document. Applications that use static UI objects will commonly respond to this message by enabling the objects.

Source: A GenDocument object.

Destination: The document object sends this message to itself.

■ MSG_GEN_DOCUMENT_ATTACH_UI_TO_DOCUMENT

```
void        MSG_GEN_DOCUMENT_ATTACH_UI_TO_DOCUMENT();
```

The document object sends this message to itself when the UI for a newly-opened document has been created. It also sends this message when re-opening a document as part of restoring GEOS from a state file. Applications may respond to this by attaching dynamic UI objects and setting the values of static UI objects.

Source: A GenDocument object.

Destination: The document object sends this message to itself.

■ MSG_GEN_DOCUMENT_DETACH_UI_FROM_DOCUMENT

```
void        MSG_GEN_DOCUMENT_DETACH_UI_FROM_DOCUMENT();
```

The document object sends this message when a document is being closed, whether because a user closes the file or because the application is being closed. It also sends this message when GEOS is in the process of saving itself to a state file prior to shutting down. Applications generally respond to this by detaching dynamic UI objects. Note that the GenDocumentControl object will automatically disable any objects in its *GDCI_enableDisableList*.

Source: A GenDocument object.

Destination: The document object sends this message to itself.

■ **MSG_GEN_DOCUMENT_DESTROY_UI_FOR_DOCUMENT**

```
void      MSG_GEN_DOCUMENT_DESTROY_UI_FOR_DOCUMENT();
```

The GenDocumentGroup object sends this message out when a document is being closed, whether because a user closes the file or because the application is being closed. Applications will generally disable static display objects and delete dynamic display objects. Note that the GenDocumentControl object will automatically disable all objects in its *GDCI_enableDisableList*, and the GenDocumentGroup will delete the display block it created for a document, if any.

Source: A GenDocument object.

Destination: The document object sends this message to itself.

■ **MSG_GEN_DOCUMENT_ATTACH_FAILED**

```
void      MSG_GEN_DOCUMENT_ATTACH_FAILED();
```

When GEOS restores itself from state, the document control tries to reattach all documents which were attached when GEOS was shut down. If this is impossible (as for example if a document was deleted after GEOS shut down), the document object will send this message to itself.

Source: A GenDocument object.

Destination: The document object sends this message to itself.

13.3.2.3 Messages Associated with Common User Actions

A single user action can precipitate several application-handled messages. This section lists the messages associated with each of several common user actions. Note that some messages are sent as the result of many user actions. If a handler needs to find out what user action caused a message to be sent, it should send MSG_GEN_DOCUMENT_GET_OPERATION to the document object.

If a message is not ordinarily handled, it is enclosed in [brackets] below. These messages are documented in the advanced usage section. Actions taken by the document control objects (other than messages sent) are listed in *italics*. The messages listed are sent by the appropriate GenDocument object to itself. If the message is not subclassed by the GenDocument object, it sends a corresponding procedural-model message (of the form

MSG_META_DOC_OUTPUT...) to the GenDocumentGroup object's output. The one exception is MSG_GEN_DOCUMENT_PHYSICAL_SAVE_AS; as noted on page 925, this message does not have a corresponding MSG_META_DOC_OUTPUT_PHYSICAL_SAVE_AS.

- New document is created:
  [MSG_GEN_DOCUMENT_PHYSICAL_CREATE]
  new file is created
  VM files: initialize VM attributes, token, protocol
  MSG_GEN_DOCUMENT_INITIALIZE_DOCUMENT_FILE
  if saveAs/Revert supported, save file so revert will return to this state
  [MSG_GEN_DOCUMENT_WRITE_CACHED_DATA_TO_FILE]
  [MSG_GEN_DOCUMENT_PHYSICAL_SAVE]
  MSG_GEN_DOCUMENT_CREATE_UI_FOR_DOCUMENT
  MSG_GEN_DOCUMENT_ATTACH_UI_TO_DOCUMENT

- Document is opened:
  [MSG_GEN_DOCUMENT_PHYSICAL_OPEN]
  [MSG_GEN_DOCUMENT_READ_CACHED_DATA_FROM_FILE]
  MSG_GEN_DOCUMENT_CREATE_UI_FOR_DOCUMENT
  MSG_GEN_DOCUMENT_ATTACH_UI_TO_DOCUMENT

- Document is saved:
  [MSG_GEN_DOCUMENT_WRITE_CACHED_DATA_TO_FILE]
  [MSG_GEN_DOCUMENT_PHYSICAL_SAVE]
  VM files: call made to VMSave

- Document is "Saved As":
  [MSG_GEN_DOCUMENT_WRITE_CACHED_DATA_TO_FILE]
  [MSG_GEN_DOCUMENT_PHYSICAL_SAVE_AS]
  [DOS files: MSG_GEN_DOCUMENT_PHYSICAL_SAVE_AS_FILE_HANDLE]
  VM files: VMSaveAs called
  [MSG_GEN_DOCUMENT_SAVE_AS_COMPLETED]

- Document is reverted to last-saved version:
  MSG_GEN_DOCUMENT_DETACH_UI_FROM_DOCUMENT
  [MSG_GEN_DOCUMENT_PHYSICAL_REVERT]
  VM: VMRevert called
  [MSG_GEN_DOCUMENT_READ_CACHED_DATA_FROM_FILE]
  MSG_GEN_DOCUMENT_ATTACH_UI_TO_DOCUMENT

- Document is about to be closed:
  [MSG_GEN_DOCUMENT_PHYSICAL_CHECK_FOR_MODIFICATIONS]

If document is modified & user wants to save changes:

[MSG_GEN_DOCUMENT_WRITE_CACHED_DATA_TO_FILE]
[MSG_GEN_DOCUMENT_PHYSICAL_SAVE]
VM: VMSave called
MSG_GEN_DOCUMENT_DETACH_UI_FROM_DOCUMENT
MSG_GEN_DOCUMENT_DESTROY_UI_FOR_DOCUMENT

If document is modified and user does not want to save changes:

MSG_GEN_DOCUMENT_DETACH_UI_FROM_DOCUMENT
[MSG_GEN_DOCUMENT_PHYSICAL_REVERT]
VM: VMRevert called
MSG_GEN_DOCUMENT_DESTROY_UI_FOR_DOCUMENT
[MSG_GEN_DOCUMENT_PHYSICAL_CLOSE]

If document is not modified:

MSG_GEN_DOCUMENT_DETACH_UI_FROM_DOCUMENT
MSG_GEN_DOCUMENT_DESTROY_UI_FOR_DOCUMENT
[MSG_GEN_DOCUMENT_PHYSICAL_CLOSE]

If document is not modified and untitled:

MSG_GEN_DOCUMENT_DETACH_UI_FROM_DOCUMENT
MSG_GEN_DOCUMENT_DESTROY_UI_FOR_DOCUMENT
[MSG_GEN_DOCUMENT_PHYSICAL_CLOSE]
[MSG_GEN_DOCUMENT_PHYSICAL_REVERT]

◆ GEOS restoring from state, document being attached:
[MSG_GEN_DOCUMENT_PHYSICAL_OPEN]
[MSG_GEN_DOCUMENT_READ_CACHED_DATA_FROM_FILE]
MSG_GEN_DOCUMENT_ATTACH_UI_TO_DOCUMENT

◆ GEOS restoring from state, attach failed:
MSG_GEN_DOCUMENT_ATTACH_FAILED
MSG_GEN_DOCUMENT_DESTROY_UI_FOR_DOCUMENT

◆ GEOS shutting down, document being detached:
[MSG_GEN_DOCUMENT_DETACH_UI_FROM_DOCUMENT]
[MSG_GEN_DOCUMENT_PHYSICAL_UPDATE]
VM files: VMUpdate called
MSG_GEN_DOCUMENT_DETACH_UI_FROM_DOCUMENT
[MSG_GEN_DOCUMENT_PHYSICAL_CLOSE]

13.4 Advanced DC Usage

By now, you should know enough for most uses of the document control. For single-document applications which manage GEOS Virtual Memory files and use a generic interface, the above documentation should be sufficient. However, there are some needs which require more advanced techniques. This section details these techniques.

This section discusses the use of document protocols to smooth the process of upgrading software. It also discusses using the document control to manage multiple documents simultaneously and to manage DOS files. Finally, it discusses those messages which an application might need to handle but would not ordinarily need to know about.

13.4.1 Document Protocols

```
MSG_META_DOC_OUTPUT_OPEN_EARLIER_COMPATIBLE_DOCUMENT,
MSG_META_DOC_OUTPUT_OPEN_EARLIER_INCOMPATIBLE_DOCUMENT,
MSG_GEN_DOCUMENT_OPEN_EARLIER_COMPATIBLE_DOCUMENT,
MSG_GEN_DOCUMENT_OPEN_EARLIER_INCOMPATIBLE_DOCUMENT
```

One difficulty in upgrading software is that an obsolete program may have created many documents. If the new version can't read those documents, people who used the old version will be inconvenienced; however, if the new versions always use the same document formats as the old versions, options for improvement will be limited. Above all, if document formats change, the new version should detect this gracefully, without crashing or damaging the old file.

The header for a GEOS Virtual Memory file contains two words for protocol numbers. The document control objects use the protocol numbers to insure that a document is compatible with the version of the application which is opening it. There are two parts to the protocol number: the *major* protocol number, and the *minor* protocol number. (If a document has a major protocol number of 3 and a minor number of 11, it is referred to has having protocol 3.11.) By convention, versions of an application with entirely compatible document formats will have the same major protocol number; if a new version of an application cannot read older documents without converting them in some way, it will have a higher major protocol number, and the minor number will be reset to zero.

When the GenDocumentGroup object opens a file, it checks the major and minor protocol numbers. It will then take appropriate action:

- If the document's major and minor protocol numbers match the protocol attributes of the GenDocumentGroup object, the document will be opened normally.
- If the document has a higher protocol number than the GenDocumentGroup (i.e. either the document has a higher major protocol number, or the document and the GenDocumentGroup have the same major protocol number and the document has a higher minor protocol number), the document control will display an appropriate alert box, after which it will close the file and delete the document object. (It will do all of this automatically, without any attention from the application.)
- If the document has lower major protocol number than the document control, the document control will send MSG_META_DOC_OUTPUT_UPDATE_EARLIER_INCOMPATIBLE_DOCUMENT (and a corresponding MSG_GEN_DOCUMENT_...). If neither message is handled, or if a handler returns an error, the document control will display an alert box and will close the file and delete the document object.
- If the document has the same major but a lower minor protocol number than the GenDocumentGroup, the document control will send MSG_META_DOC_OUTPUT_UPDATE_EARLIER_COMPATIBLE_DOCUMENT (and a corresponding MSG_GEN_DOCUMENT_...). After this, it will proceed normally, whether the messages were handled or not (since the document is presumed to be compatible). If a handler returns an error, it will close the file and free the document object.

Note that the document control will not change the file under any of these circumstances. In particular, if it opens an earlier document, it will not change the document's protocol number. If the application wishes to do this, it should do it explicitly (generally in the handlers for the "UPDATE_..._DOCUMENT" messages). The protocol numbers are among a file's extended attributes. For information about changing extended attributes, see section 17.5.3 of "File System," Chapter 17 of the Concepts Book.

■ MSG_META_DOC_OUTPUT_UPDATE_EARLIER_COMPATIBLE_DOCUMENT

```
Boolean  MSG_META_DOC_OUTPUT_UPDATE_EARLIER_COMPATIBLE_DOCUMENT(
         word *            error,    /* Return error code from FileError */
         optr              document, /* pointer to document object */
         FileHandle        file);    /* handle of file opened */
```

The GenDocumentGroup object sends this message to its output when the user tries to open a document with the same major protocol number as the document control and a lower minor protocol number. Applications will commonly respond to this message by changing the document's protocol number to bring it up-to-date. If the application can't use the document, it should return *true* and set **error*. (File access error codes are members of the **FileError** enumerated type, defined in **file.h**.) With an error, the document control will close the document unchanged. If the application successfully updates the document, it should return zero and set **error* to zero.

Source: The GenDocumentGroup object.

Destination: Output of GenDocumentGroup (usually the Process object)

Parameters:

error	A pointer to a word in which an error should be returned.
document	The optr of the appropriate document object.
file	The FileHandle of the appropriate file.

Return: *true* if error occurs.

**error*	**FileError** code (or zero if there is no error).

Interception: You must write a handler for this message in whatever class will be receiving it (usually the process class).

■ MSG_GEN_DOCUMENT_UPDATE_EARLIER_COMPATIBLE_DOCUMENT

```
Boolean  MSG_GEN_DOCUMENT_UPDATE_EARLIER_COMPATIBLE_DOCUMENT(
         word * error); /* Return error code from FileError type */
```

The document object sends this message when the user tries to open a document with the same major protocol number as the document control and a lower minor protocol number. Applications will commonly respond to this message by changing the document's protocol number to bring it up-to-date. If the application can't use the document, it should return *true* and put an error code in **error*. (File access error codes are members of the **FileError** enumerated type, defined in **file.h**.) With an error, the document control will close the document unchanged. If the application successfully updates the document, it should return zero and set **error* to zero.

Source: A GenDocument object.

Destination: The document object sends this message to itself.

Parameters: *error* A pointer to a word in which an error should be returned.

Return: *true* if error occurs.

error* **FileError code (or zero if there is no error).

■ MSG_META_DOC_OUTPUT_UPDATE_EARLIER_INCOMPATIBLE_DOCUMENT

```
Boolean MSG_META_DOC_OUTPUT_UPDATE_EARLIER_INCOMPATIBLE_DOCUMENT(
                                                /* Return true if error */
        word *error, /* Return error code from FileError enum. type */
        optr document, /* pointer to document object */
        FileHandle file); /* handle of file opened */
```

The GenDocumentGroup object sends this message to its output when the user tries to open a document with a lower major protocol number than the document control. Applications respond to the message by making any changes to the document necessary to make it compatible with the application. The application should also change the document's protocol numbers. If the application can't use the document, it should return *true* and put an error code in **error*. (File access error codes are members of the **FileError** enumerated type, defined in **file.h**.) With an error, the document control will close the document unchanged. If the application successfully updates the document, it should return zero and set **error* to zero.

Source: The GenDocumentGroup object.

Destination: Output of GenDocumentGroup (usually the Process object).

Parameters: *error* A pointer to a word in which an error should be returned.

document The optr of the appropriate document object.

file The FileHandle of the appropriate file.

Return: *true* if error occurs.

error* **FileError code (or zero if there is no error).

Interception: You must write a handler for this message in whatever class will be receiving it (usually the process class)

■ MSG_GEN_DOCUMENT_UPDATE_EARLIER_INCOMPATIBLE_DOCUMENT

```
Boolean  MSG_GEN_DOCUMENT_UPDATE_EARLIER_INCOMPATIBLE_DOCUMENT(
         word * error); /* Return error code from FileError type */
```

13.4

The document object sends this message when the user tries to open a document with a lower major protocol number than the document control. Applications respond to this message by making any changes to the document necessary to make it compatible with the application. The application should also change the document's protocol numbers. If the application can't use the document, it should return *true* and put an error code in **error*. (File access error codes are members of the **FileError** enumerated type, defined in **file.h**.) With an error, the document control will close the document unchanged. If the application successfully updates the document, it should return zero and set **error* to zero.

Source: A GenDocument object.

Destination: The document object sends this message to itself.

Parameters: *error* — A pointer to a word in which an error should be returned.

Return: *true* if error occurs.

error* — **FileError code (or zero if there is no error).

13.4.2 Multiple Document Model

The Object model of document control makes it easy to manage several documents at once. Effectively, each document object acts as a special-purpose application dedicated to handling one file. The display control lets the user change documents at will. The document control and display control take care of most of the nuts and bolts of file switching, so the application doesn't have to worry about them.

13.4.2.1 Subclassing GenDocumentClass

Under the object model of document control, a document's functionality is implemented as methods in the document class. Much of the switching between documents is thus transparent to the application. When the target changes, all user actions will result in messages being sent to the new target document; whenever a document gets a message, it knows the message pertains to itself, not some other document (and thus knows that it is the target document).

In order to implement this functionality, the application must declare a subclass of **GenDocumentClass**. This subclass will have its own methods for application-handled messages.

13.4.2.2 Using the Display Group

The simplest way to manage multiple documents is to use the display control. The display control lets the user change documents transparently to the application. In order to do this, the application must define a resource of objects which should be copied each time a document is opened or created. The GenDocumentGroup object's *GDGI_genDisplay* attribute should be set to point to a GenDisplay object in that resource; the GenDisplay should be set "not usable" and should be the head of a tree of objects. Also, the application should define an object of **GenDisplayGroupClass**, and the GenDocumentGroup's *GDGI_genDisplayGroup* attribute should be set to point to it. The display control will then automatically switch displays whenever the user chooses an entry from the display control.

13.4.3 Working with DOS files

```
MSG_META_DOC_OUTPUT_PHYSICAL_SAVE,
MSG_GEN_DOCUMENT_PHYSICAL_SAVE,
MSG_META_DOC_OUTPUT_PHYSICAL_UPDATE,
MSG_GEN_DOCUMENT_PHYSICAL_UPDATE,
MSG_META_DOC_OUTPUT_PHYSICAL_SAVE_AS_FILE_HANDLE,
MSG_GEN_DOCUMENT_PHYSICAL_SAVE_AS_FILE_HANDLE,
```

MSG_META_DOC_OUTPUT_PHYSICAL_REVERT,
MSG_GEN_DOCUMENT_PHYSICAL_REVERT

The document control can be used to handle DOS files. However, there are special issues to be aware of. When you use GEOS Virtual Memory files, the system takes care of swapping sections of the file in and out of memory as needed. You can use high-level commands to mark parts of the file as dirty, and when you need the document saved, only the dirty sections will be copied to the disk. The details of reading from and writing to the disk are transparent to the application.

When you use DOS files, on the other hand, you have to take care of all of these details yourself. It is usually impractical to keep all of a document in memory at one time, so you have to have some way of managing the data (perhaps by creating a temporary VM file and copying the DOS file into that).

For this reason, the document control sends out messages when it does many low-level things (such as save files). If the application needs to take special actions, it can define handlers for these messages. Most of these messages can be ignored if you are working with GEOS files.

If you want to implement “Save As” and “Revert” for DOS files, you will have to do most of it by hand. If you leave “Save As” and “Revert” enabled, the Document Control will do some of the work for you. For example, when the user chooses “Save As”, the Document Control will first present a File Selector, letting the user choose a file name and location. The Document Control will then create the new file. After this it will send out MSG_META_DOC_OUTPUT_PHYSICAL_SAVE_AS_FILE_HANDLE and MSG_GEN_DOCUMENT_PHYSICAL_SAVE_AS_FILE_HANDLE, passing the handle of the newly-created file. The application is responsible for writing the current version of the document to the new file, and reverting the original file to its last-saved state. The Document Control will automatically close the original file and update all Document Control instance data as necessary.

■ MSG_META_DOC_OUTPUT_PHYSICAL_SAVE

```
Boolean  MSG_META_DOC_OUTPUT_PHYSICAL_SAVE(
         word *              error,
         optr                document,
         FileHandle          file);
```

If you need to take special steps to save a file, you should have a handler for either this message or MSG_GEN_DOCUMENT_PHYSICAL_SAVE. The handler should write the file completely to the disk. If an error occurs, return *true* and write the error code in **error*. (File access error codes are members of the **FileError** enumerated type, defined in **file.h**.)

If, for example, you copy a DOS file into a temporary VM file while you work on it, you would probably respond to this message by copying the data from the temporary file back to the DOS file.

Source: The GenDocumentGroup object.

Destination: Output of GenDocumentGroup (usually the Process object).

Parameters: *error* A pointer to a word in which an error should be returned.

document The optr of the appropriate document object.

file The FileHandle of the appropriate file.

Interception: DOS-based applications must handle either this message or MSG_GEN_DOCUMENT_PHYSICAL_SAVE. Applications which use GEOS data files will generally not intercept this message.

Return: *true* if error occurs.

error* **FileError code (or zero if there is no error).

■ MSG_GEN_DOCUMENT_PHYSICAL_SAVE

```
Boolean MSG_GEN_DOCUMENT_PHYSICAL_SAVE(
        word * error);          /* Error code from FileError type */
```

This message is sent when the user saves a file. If you need to take special steps to save a file, you should have a handler for either this message or MSG_META_DOC_OUTPUT_PHYSICAL_SAVE. The handler should write the file completely to the disk. If an error occurs, return *true* and write the error code in **error*. (File access error codes are members of the **FileError** enumerated type, defined in **file.h**.)

If, for example, you copy a DOS file into a temporary VM file while you work on it, you would probably respond to this message by copying the data from the temporary file back to the DOS file.

Source: The GenDocument object.

Destination: The document object sends this message to itself.

Parameters: *error* A pointer to a word in which an error should be returned.

Return: *true* if error occurs.

error* **FileError code (or zero if there is no error).

Interception: DOS-based applications must handle either this message or MSG_META_DOC_OUTPUT_PHYSICAL_SAVE. Applications which use GEOS data files will generally not intercept this message.

■ MSG_META_DOC_OUTPUT_PHYSICAL_UPDATE

13.4

```
Boolean  MSG_META_DOC_OUTPUT_PHYSICAL_UPDATE(occurred */
         word *            error,    /* Error code from FileError type */
         optr              document, /* Pointer to document object */
         FileHandle        file);    /* Handle of DOS file */
```

This message is sent when the file is auto-saved (if this is enabled), and when the document is detached as part of a GEOS shutdown. If you need to take special steps to save a file, you should have a handler for either this message or MSG_GEN_DOCUMENT_PHYSICAL_UPDATE. The handler should write the file completely to the disk. If an error occurs, return *true* and write the error code in **error*. (File access error codes are members of the **FileError** enumerated type, defined in **file.h**.)

If, for example, you copy a DOS file into a temporary VM file while you work on it, you would probably respond to this message by copying the data from the temporary file back to the DOS file.

Source: The GenDocumentGroup object.

Destination: Output of GenDocumentGroup (usually the Process object).

Parameters:

error A pointer to a word in which an error should be returned.

document The optr of the appropriate document object.

file The FileHandle of the appropriate file.

Return: *true* if error occurs.

error* **FileError code (or zero if there is no error).

Interception: DOS-based applications which will have auto-save capability must handle either this message or MSG_GEN_DOCUMENT_PHYSICAL_UPDATE. Applications which use GEOS data files will generally not intercept this message.

■ MSG_GEN_DOCUMENT_PHYSICAL_UPDATE

```
Boolean    MSG_GEN_DOCUMENT_PHYSICAL_UPDATE(
           word * error);         /* Error code from FileError type */
```

This message is sent when the file is auto-saved (if this is enabled), and when the document is detached as part of a GEOS shutdown. If you need to take special steps to save a file, you should have a handler for either this message or MSG_META_DOC_OUTPUT_PHYSICAL_UPDATE. The handler should write the file completely to the disk. If an error occurs, return *true* and write the error code in **error*. (File access error codes are members of the **FileError** enumerated type, defined in **file.h**.)

13.4

If, for example, you copy a DOS file into a temporary VM file while you work on it, you would probably respond to this message by copying the data from the temporary file back to the DOS file.

Source: A GenDocument object.

Destination: The document object sends this message to itself.

Parameters: *error* A pointer to a word in which an error should be returned.

Return: *true* if error occurs.

error* **FileError code (or zero if there is no error).

Interception: DOS-based applications which will have auto-save capability must handle either this message or MSG_META_DOC_OUTPUT_PHYSICAL_UPDATE. Applications which use GEOS data files will generally not intercept this message.

■ MSG_META_DOC_OUTPUT_PHYSICAL_SAVE_AS_FILE_HANDLE

```
Boolean    MSG_META_DOC_OUTPUT_PHYSICAL_SAVE_AS_FILE_HANDLE(
           word *              error,
           optr                document,
           FileHandle          file);
```

This message is sent when the Document Control is ready to “save-as” a DOS file. The Document Control will have asked the user what the new file should be, and will have created an appropriate file. The handler for this message must write the current version of the document to the new file, and restore the original file to its last-saved state.

Source: The GenDocumentGroup object.

Destination: The output of GenDocumentGroup (usually the Process object).

Parameters: *error* A pointer to a word in which an error code should be returned.

document The optr of the appropriate document object.

file The handle of the newly-opened file. The current version of the document should be saved to this file.

Return: *true* if an error occurred.

file* A member of the **FileError enumerated type (if an error occurred).

Interception: DOS-file applications must intercept this message (or MSG_GEN_DOCUMENT_PHYSICAL_SAVE_AS_FILE_HANDLE) if they wish to implement save-as/revert functionality.

■ MSG_GEN_DOCUMENT_PHYSICAL_SAVE_AS_FILE_HANDLE

```
Boolean  MSG_GEN_DOCUMENT_PHYSICAL_SAVE_AS_FILE_HANDLE(
         word *              error,
         FileHandle          file);
```

This message is sent when the Document Control is ready to “save-as” a DOS file. The Document Control will have asked the user what the new file should be, and will have created an appropriate file. The handler for this message must write the current version of the document to the new file, and restore the original file to its last-saved state.

Source: A GenDocument.

Destination: The GenDocument object sends this message to itself.

Parameters: *error* A pointer to a word in which an error code should be returned.

file The handle of the newly-opened file. The current version of the document should be saved to this file.

Return: *true* if an error occurred.

error* A member of the **FileError enumerated type (if an error occurred).

Interception: DOS-file applications must intercept this message (or MSG_META_DOC_OUTPUT_PHYSICAL_SAVE_AS_FILE_HANDLE) if they wish to implement save-as/revert functionality.

■ MSG_META_DOC_OUTPUT_PHYSICAL_REVERT

```
Boolean MSG_META_DOC_OUTPUT_PHYSICAL_REVERT(
        word *              error,
        optr                document,
        FileHandle          file);
```

The Document Control sends this message to revert a DOS file to its last-saved state. The handler must restore the file to its condition as of the last time it was saved.

Source: The GenDocumentGroup object.

Destination: The output of GenDocumentGroup (usually the Process object).

Parameters: *error* A pointer to a word in which an error should be returned.

document The optr of the appropriate document object.

file The FileHandle of the appropriate file.

Return: *true* if error occurs.

error* **FileError code (or zero if there is no error).

Interception: DOS-file applications must intercept this message (or MSG_GEN_DOCUMENT_PHYSICAL_REVERT) if they wish to implement save-as/revert functionality.

■ MSG_GEN_DOCUMENT_PHYSICAL_REVERT

```
Boolean MSG_GEN_DOCUMENT_PHYSICAL_REVERT(
        word *              error,
        FileHandle          file);
```

The Document Control sends this message to revert a DOS file to its last-saved state. The handler must restore the file to its condition as of the last time it was saved.

Source: The GenDocumentGroup object.

Destination: The output of GenDocumentGroup (usually the Process object).

Parameters: *error* A pointer to a word in which an error should be returned.

file The FileHandle of the appropriate file.

Return: *true* if error occurs.

error* **FileError code (or zero if there is no error).

Interception: DOS-file applications must intercept this message (or MSG_META_DOC_OUTPUT_PHYSICAL_REVERT) if they wish to implement save-as/revert functionality.

13.4.4 Special-Purpose Messages

13.4

In addition to the basic messages discussed above, there are messages the document control sends out which do not ordinarily need to be handled. Some of these messages have been discussed above; most of the rest are described here.

13.4.4.1 Caching Data in Memory

```
MSG_META_DOC_OUTPUT_WRITE_CACHED_DATA_TO_FILE,
MSG_GEN_DOCUMENT_WRITE_CACHED_DATA_TO_FILE,
MSG_META_DOC_OUTPUT_READ_CACHED_DATA_FROM_FILE,
MSG_GEN_DOCUMENT_READ_CACHED_DATA_FROM_FILE
```

Sometimes an application will want to keep frequently-accessed data in memory. For example, if you are managing Virtual Memory files, you may want to copy the map block to a fixed memory block instead of locking the block every time you need to read or change it. This is known as *caching* data.

If you cache data, you must make sure that the application's version of the data is consistent with the disk file. The document control helps keep track of this. Whenever the file (or the state) is saved, the document control will first send a message instructing the application to write the cache to the file, then it will save the file. Similarly, when the file is opened or GEOS is restarted from state, the document control will send a message instructing the application to reload the cached data from the file.

There is one special concern. The user cannot save a file unless it has been marked dirty; also, the document control does not send MSG_..._WRITE_CACHED_DATA_TO_FILE to documents which are not dirty. Therefore, if you change the data cache without actually altering the file, you should send a MSG_GEN_DOCUMENT_GROUP_MARK_DIRTY to the GenDocumentGroup.

■ MSG_META_DOC_OUTPUT_READ_CACHED_DATA_FROM_FILE

```
void      MSG_META_DOC_OUTPUT_READ_CACHED_DATA_FROM_FILE(
          optr                document, /* optr of document object */
          FileHandle          file);    /* FileHandle of associated file */
```

The GenDocumentGroup sends this message when the document needs to read cached data. In particular, it sends this when a document is opened, when a document is reverted to its last-saved state, and when a document is re-opened as GEOS restores from state. If the application maintains a data cache, it should read the data from the file at this point. If the document does not cache data, it can ignore this message.

Note that if the document control notices that the file has changed on disk, it will not send this message; it will, however, send a MSG_META_DOC_OUTPUT_DOCUMENT_HAS_CHANGED. The handler for that message should reread the cache or call the handler for this message.

Source: The GenDocumentGroup object.

Destination: Output of GenDocumentGroup (usually the Process object).

Parameters: *document* *The optr of the appropriate document object.*

file The FileHandle of the appropriate file.

Interception: You must write a handler for this message in whatever class will be receiving it (usually the process class).

■ MSG_GEN_DOCUMENT_READ_CACHED_DATA_FROM_FILE

```
void      MSG_GEN_DOCUMENT_READ_CACHED_DATA_FROM_FILE();
```

The document object sends this message to itself when the document needs to read cached data. In particular, it sends this when a document is opened, when a document is reverted to its last-saved state, and when a document is re-opened as GEOS restores from state. If the application maintains a data cache, it should read the data from the file at this point. If the document does not cache data, it can ignore this message.

Note that if the document control notices that the file has changed on disk, it will not send this message; it will, however, send a MSG_GEN_DOCUMENT_DOCUMENT_HAS_CHANGED. The handler for that message should reread the cache or call the handler for this message.

Source: A GenDocument object.

Destination: The document object sends this message to itself.

■ MSG_META_DOC_OUTPUT_WRITE_CACHED_DATA_TO_FILE

```
void      MSG_META_DOC_OUTPUT_WRITE_CACHED_DATA_TO_FILE(
          optr               document, /* optr of document object */
          FileHandle         file);    /* FileHandle of associated file */
```

The GenDocumentGroup object sends this message when the document needs to write cached data back to the file. In particular, it sends this message just before a document is saved, auto-saved, or “Saved As,” and before the document is closed as GEOS shuts down. The document should write its cached data back to the file. If the document does not cache data, it can ignore this message.

Warnings: This message will not be sent if the document is clean. Therefore, if you change the data cache without changing the file, you should send a MSG_GEN_DOCUMENT_GROUP_MARK_DIRTY_BY_FILE to the GenDocumentGroup object.

Source: The GenDocumentGroup object.

Destination: Output of GenDocumentGroup (usually the Process object)

Parameters: *document* The optr of the appropriate document object.

file The FileHandle of the appropriate file.

Interception: You must write a handler for this message in whatever class will be receiving it (usually the process class).

■ MSG_GEN_DOCUMENT_WRITE_CACHED_DATA_TO_FILE

```
void      MSG_META_DOC_OUTPUT_WRITE_CACHED_DATA_TO_FILE();
```

The document object sends this message to itself when the document needs to write cached data back to the file. In particular, it sends this message just before a document is saved, auto-saved, or “Saved As,” and before the document is closed as GEOS shuts down. The document should write its cached data back to the file. If the document does not cache data, it can ignore this message.

Warnings: This message will not be sent if the document is clean. Therefore, if you change the data cache without changing the file, you should send a MSG_GEN_DOCUMENT_GROUP_MARK_DIRTY_BY_FILE to the GenDocumentGroup object.

Source: A GenDocument object.

Destination: The document object sends this message to itself.

13.4.4.2 Other Messages To Know About

There are a few other messages which are worth knowing about. These messages alert the application to special situations. Most applications can ignore these messages; however, for a few, these messages should be handled.

■ MSG_META_DOC_OUTPUT_SAVE_AS_COMPLETED

```
void    MSG_META_DOC_OUTPUT_SAVE_AS_COMPLETED(
        optr                document, /* optr of document object */
        FileHandle          file);    /* FileHandle of associated file */
```

The GenDocumentGroup object sends this message when a "Save As" operation has been successfully completed.

Source: The GenDocumentGroup object.

Destination: Output of GenDocumentGroup (usually the Process object).

Parameters: *document* The optr of the appropriate document object.

file The FileHandle of the appropriate file.

Interception: You must write a handler for this message in whatever class will be receiving it (usually the process class).

■ MSG_GEN_DOCUMENT_SAVE_AS_COMPLETED

```
void    MSG_GEN_DOCUMENT_SAVE_AS_COMPLETED();
```

The document object sends this message to itself when a "Save As" operation has been successfully completed.

Source: A GenDocument object.

Destination: The document object sends this message to itself.

■ MSG_META_DOC_OUTPUT_DOCUMENT_HAS_CHANGED

```
void    MSG_META_DOC_OUTPUT_DOCUMENT_HAS_CHANGED(
        optr                document, /* optr of document object */
        FileHandle          file);    /* FileHandle of associated file */
```

If the GDGA_AUTOMATIC_CHANGE_NOTIFICATION attribute of the GenDocumentGroup object is set to on, a timer will periodically check to see if any open documents have been changed by another application. If they have, the GenDocumentGroup object will send this message out. The application should respond by redisplaying the data on the screen and rereading any cached data from the file.

Source: The GenDocumentGroup object.

Destination: Output of GenDocumentGroup (usually the Process object).

Parameters: *document* The optr of the appropriate document object.

file The FileHandle of the appropriate file.

Interception: You must write a handler for this message in whatever class will be receiving it (usually the process class).

■ MSG_GEN_DOCUMENT_DOCUMENT_HAS_CHANGED

```
void        MSG_GEN_DOCUMENT_DOCUMENT_HAS_CHANGED();
```

If the GDGA_AUTOMATIC_CHANGE_NOTIFICATION attribute of the GenDocumentGroup object is set to on, a timer will periodically check to see if any open documents have been changed by another application. If they have, the document object will send this message out. The application should respond by redisplaying the data on the screen and rereading any cached data from the file.

Source: A GenDocument object.

Destination: The document object sends this message to itself.

13.4.5 Forcing Actions

```
MSG_GEN_DOCUMENT_CONTROL_DISPLAY_DIALOG,
MSG_GEN_DOCUMENT_CONTROL_INITIATE_NEW_DOC,
MSG_GEN_DOCUMENT_CONTROL_INITIATE_USE_TEMPLATE_DOC,
MSG_GEN_DOCUMENT_CONTROL_INITIATE_USE_TEMPLATE_DOC,
MSG_GEN_DOCUMENT_CONTROL_INITIATE_OPEN_DOC,
MSG_GEN_DOCUMENT_CONTROL_INITIATE_IMPORT_DOC,
MSG_GEN_DOCUMENT_CONTROL_INITIATE_SAVE_DOC,
MSG_GEN_DOCUMENT_CONTROL_INITIATE_SAVE_AS_DOC,
MSG_GEN_DOCUMENT_CONTROL_INITIATE_SAVE_AS_TEMPLATE_DOC,
MSG_GEN_DOCUMENT_CONTROL_INITIATE_COPY_TO_DOC,
MSG_GEN_DOCUMENT_CONTROL_INITIATE_EXPORT_DOC,
```

MSG_GEN_DOCUMENT_CONTROL_INITIATE_SET_TYPE_DOC,
MSG_GEN_DOCUMENT_CONTROL_INITIATE_SET_PASSWORD_DOC(

If you wish, you can force the document control to take certain actions as if the user had requested them. You do this by sending the message which would ordinarily trigger such an action. For example, when the user selects the "save" trigger, that trigger sends the message MSG_GEN_DOCUMENT_CONTROL_INITIATE_SAVE_DOC to the Document Control object. If you wish, you can force a save by sending this message manually; the document control will behave as if the user had selected that action.

■ MSG_GEN_DOCUMENT_CONTROL_DISPLAY_DIALOG

```
void        MSG_GEN_DOCUMENT_CONTROL_DISPLAY_DIALOG();
```

This message forces the document control to display the opening "New/Open" dialog box, as if the user had selected the "New/Open" trigger on the File menu.

Source: Unrestricted.

Destination: The GenDocumentControl object.

Parameters: None.

Interception: This message is not normally intercepted.

■ MSG_GEN_DOCUMENT_CONTROL_INITIATE_NEW_DOC

```
void        MSG_GEN_DOCUMENT_CONTROL_INITIATE_NEW_DOC();
```

This message forces the document control to create a new file, exactly as if the user had requested it.

Source: Unrestricted.

Destination: The GenDocumentControl object.

Parameters: None.

Interception: This message is not normally intercepted.

13.4

■ MSG_GEN_DOCUMENT_CONTROL_INITIATE_USE_TEMPLATE_DOC

```
void        MSG_GEN_DOCUMENT_CONTROL_INITIATE_USE_TEMPLATE_DOC();
```

This message forces the document control to create a new file from a template (bringing up an appropriate file selector), exactly as if the user had requested it.

Source: Unrestricted.

Destination: The GenDocumentControl object.

Parameters: None.

Interception: This message is not normally intercepted.

■ MSG_GEN_DOCUMENT_CONTROL_INITIATE_OPEN_DOC

```
void        MSG_GEN_DOCUMENT_CONTROL_INITIATE_OPEN_DOC();
```

This message forces the document control to open a file (bringing up the appropriate file selector), exactly as if the user had requested it.

Source: Unrestricted.

Destination: The GenDocumentControl object.

Parameters: None.

Interception: This message is not normally intercepted.

■ MSG_GEN_DOCUMENT_CONTROL_INITIATE_IMPORT_DOC

```
void        MSG_GEN_DOCUMENT_CONTROL_INITIATE_IMPORT_DOC();
```

This message forces the document control to import a file (bringing up the appropriate file selector), exactly as if the user had requested it.

Source: Unrestricted.

Destination: The GenDocumentControl object.

Parameters: None.

Interception: This message is not normally intercepted.

■ MSG_GEN_DOCUMENT_CONTROL_INITIATE_SAVE_DOC

```
void        MSG_GEN_DOCUMENT_CONTROL_INITIATE_SAVE_DOC();
```

This message forces the document control to save the active file, exactly as if the user had requested it.

Source: Unrestricted.

Destination: The GenDocumentControl object.

Parameters: None.

Interception: This message is not normally intercepted.

■ MSG_GEN_DOCUMENT_CONTROL_INITIATE_SAVE_AS_DOC

```
void	MSG_GEN_DOCUMENT_CONTROL_INITIATE_SAVE_AS_DOC();
```

This message forces the document control to save the active file under a new name (bringing up the appropriate file selector), exactly as if the user had requested it.

Source: Unrestricted.

Destination: The GenDocumentControl object.

Parameters: None.

Interception: This message is not normally intercepted.

■ MSG_GEN_DOCUMENT_CONTROL_INITIATE_SAVE_AS_TEMPLATE_DOC

```
void	MSG_GEN_DOCUMENT_CONTROL_INITIATE_SAVE_AS_TEMPLATE_DOC();
```

This message forces the document control to save the active file as a template (bringing up the appropriate file selector), exactly as if the user had requested it.

Source: Unrestricted.

Destination: The GenDocumentControl object.

Parameters: None.

Interception: This message is not normally intercepted.

■ MSG_GEN_DOCUMENT_CONTROL_INITIATE_COPY_TO_DOC

```
void	MSG_GEN_DOCUMENT_CONTROL_INITIATE_COPY_TO_DOC();
```

This message forces the document control to copy the active file to a new name (bringing up the appropriate file selector), exactly as if the user had requested it.

Source: Unrestricted.

Destination: The GenDocumentControl object.

Parameters: None.

Interception: This message is not normally intercepted.

■ MSG_GEN_DOCUMENT_CONTROL_INITIATE_EXPORT_DOC

```
void    MSG_GEN_DOCUMENT_CONTROL_INITIATE_EXPORT_DOC();
```

This message forces the document control to export the active file (bringing up the appropriate file selector), exactly as if the user had requested it.

Source: Unrestricted.

Destination: The GenDocumentControl object.

Parameters: None.

Interception: This message is not normally intercepted.

■ MSG_GEN_DOCUMENT_CONTROL_INITIATE_SET_TYPE_DOC

```
void    MSG_GEN_DOCUMENT_CONTROL_INITIATE_SET_TYPE_DOC();
```

This message forces the document control to change the type (public, read-only, etc.) of the active file (bringing up the appropriate dialog box), exactly as if the user had requested it.

Source: Unrestricted.

Destination: The GenDocumentControl object.

Parameters: None.

Interception: This message is not normally intercepted.

■ MSG_GEN_DOCUMENT_CONTROL_INITIATE_SET_PASSWORD_DOC

```
void    MSG_GEN_DOCUMENT_CONTROL_INITIATE_SET_PASSWORD_DOC();
```

This message forces the document control to set the password of the active file (bringing up the appropriate dialog box), exactly as if the user had requested it.

Source: Unrestricted.

Destination: The GenDocumentControl object.

Parameters: None.

Interception: This message is not normally intercepted.

GenFile Selector

14

The GenFileSelector provides the user interface and functionality to allow a user to traverse directories and volumes (disks) and view the files contained therein. It can be customized to limit the displayed file names or to expand the listings to include special files.

This object is used by the GenDocumentControl object to provide most file dialog boxes. If your application uses the Document Control objects, you will probably not need to add a GenFileSelector.

You should be familiar with user interface objects in general before reading this chapter. You should also have an understanding of file names, GEOS files, disk handles, and the DOS directory structure. For this information, see both “GenDocument,” Chapter 13 of this book, and “File System,” Chapter 17 of the Concepts Book.

14.1 File Selector Overview

When you are not using the Document Control objects but need to allow the user to traverse directories and disks in order to locate a given file, you should use a GenFileSelector. The File Selector is designed to give tremendous flexibility for many different types of file searches, providing both directory scanning and user interface. An OSF/Motif File Selector is shown in Figure 14-1.

The File Selector allows users to browse all the directories and disks readable by DOS, including readable network drives. This browsing is completely internal to the File Selector; until the user selects a file from the list, your application does not need to know what the File Selector is doing.

You can customize the searches the File Selector makes in several different ways:

- Setting paths, volumes, and selections
  You may direct the File Selector to show any given directory on any given volume, and you may also set the selected file displayed. Additionally,

you can navigate directories with messages sent to the File Selector. See section 14.4.2 on page 979.

- Limiting a directory scan to certain files
  The File Selector can limit its directory scans to show only the files you request. You may limit the scan based on file types, file attributes, or filename extensions. See section 14.3 on page 951.
- Searching for associated files
  You can limit the File Selector's searches to only those files that match a given creator or token. That is, the File Selector will display only files created by a particular geode. See section 14.3.3 on page 959.
- Customizing the searches
  If the available search options don't fit your needs, you can create a customized search. If this type of search is employed, the File Selector will send a special message to itself—to take advantage of this, you can easily subclass the GenFileSelector and write a custom handler for this special message. See section 14.3.8 on page 972.

All of the directory scanning and path navigation is internal to the GenFileSelector. When a selection is made (a user single- or double-clicks on an entry, for example), the File Selector will send out a message indicating the selection. You can, however, force the File Selector to change to any directory or search criteria at any time.

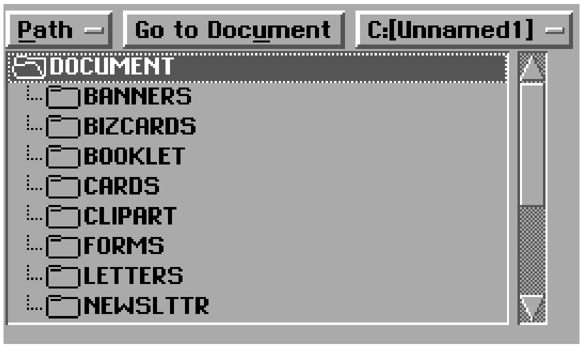


Figure 14-1 *A GenFileSelector*

The simple configuration shown will normally be supplemented by a Reply Bar and placed in a dialog box.

◆Objects

14.2 File Selector Basics

The simplest File Selector configuration is shown in Figure 14-1. Nearly all applications, however, will use a configuration similar to that shown in Figure 14-2. The following sections describe how to set up and use the File Selector in this configuration and how to modify the configuration for common uses.

14.2

Most File Selectors will be put in dialog boxes like that shown in Figure 14-2 and will be accompanied by at the least a Reply Bar. (In the figure, the “Open,” “Cancel,” and “?” buttons comprise the Reply Bar. For more information on Reply Bars in general, see “GenInteraction,” Chapter 7.) Typically, the dialog box will be brought up by a menu item such as *Open...* or *Insert From Text File...*.

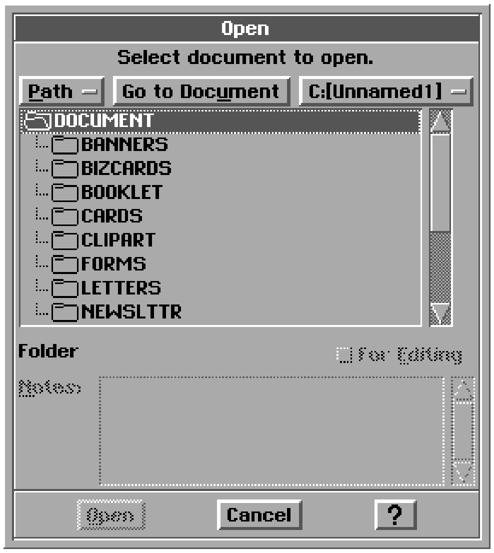


Figure 14-2 *An Open File Selector for GeoWrite*

Generally, a File Selector is given a Reply Bar and in some cases other objects for added functionality.

Figure 14-2 shows a sample *Insert From Text File* dialog box. (Note that you would probably use the Impex object for this functionality; this is provided for example.) This box has three basic elements: The GenGlyphDisplay (the title at the top), the File Selector (the scrolling list of files and the disk and folder icons), and the Reply Bar. In nearly all cases, these three elements will be included with every File Selector in its dialog box.

14.2.1 Setting Up the File Selector

To set up the File Selector as shown in Figure 14-2, you will have to set up a GenInteraction as a dialog box. (For information on menus and dialog boxes, see “GenInteraction,” Chapter 7.) The File Selector and other elements of the dialog should be placed as children of the Interaction.

Code Display 14-1 shows the Goc code required to display and use a File Selector in this basic form. You can use this code to create a similar dialog box with the proper elements or modify it to gain the functionality you want. Specific permutations of this configuration will be described in section 14.2.4 on page 949; however, the remainder of this section describes the basics of the File Selector.

Code Display 14-1 A Basic File Selector

```
/* Extra code for menus and other UI objects is left out. Only what is
 * required for these objects is included. */

/* This dialog box provides the organization needed to contain the File
 * Selector, the name of the box, and the Reply Bar. */
@object GenInteractionClass MyDialogBox = {
    GI_visMoniker = ‘I’, “Insert From Text File”;
    GI_comp = MyGlyph, MyFileSel, MyInsertTrigger;
    GII_visibility = (GIV_DIALOG);
    GII_type = (GIT_COMMAND);
};

/* The Glyph Display provides the label in the dialog box. Alternatively, the
 * moniker of the GenInteraction could be used for this functionality. */
@object GenGlyphClass MyGlyph = {
    GI_visMoniker = “Insert From Text File”;
}
```

```
/* The File Selector is as basic as possible. */
@object GenFileSelectorClass MyFileSel = {
    GFSI_destination = process;          /* The object receiving notification. */
    GFSI_notificationMsg = MSG_MY_APP_FILE_SELECTED;     /* The message sent
                                                           * upon selection. */
};

@object GenTriggerClass MyInsertTrigger = {
    GI_visMoniker = “Insert”;
    GTI_destination = process;
    GTI_actionMsg = MSG_MY_APP_INSERT_TRIGGER_SELECTED;
    HINT_SEEK_REPLY_BAR;
};

/* Note that in most cases, this trigger will not be needed. Generally, the File
 * Selector will include an “OK” or “Use This File” trigger in its reply bar that
 * executes an IC_OK function. For your trigger above to demonstrate that
 * functionality, remove the GTI_… fields and add the line
 * ATTR_GEN_TRIGGER_INTERACTION_COMMAND = (IC_OK);
 */
```

14.2.2 Supporting the File Selector

To use the File Selector as shown in Code Display 14-1, your application must have some object (which may be the Process object) that can handle all the cases that may arise from a user’s interaction with the dialog box. All of these cases are listed below:

- The user clicks once on an entry in the File Selector’s list.
When this happens, the File Selector highlights the selected entry and sends its notification message to its destination object. With the message will come a record of flags indicating the action was a single click.

- The user double-clicks on an entry in the File Selector’s list.
This action causes the File Selector to send its notification message to its destination object. The default action for the dialog box containing the File Selector will also occur (e.g. the default action is what happens when the “Insert” trigger is pressed). If the current selection is a directory, the application does not need to do anything; the File Selector will open the directory automatically. If the current selection is a file, the application

must deal with the file properly and then close the dialog box containing the File Selector.

- The user clicks on the "Insert" trigger.
  This action causes the trigger to send its notification message to its destination object. In most cases, the trigger with be an IC_OK trigger that will cause the File Selector to act as if a double-click had occurred. If the current selected entry in the list is a directory, the application should instruct the File Selector to open the directory and display its file list. If the current selection is a file, the application is responsible for dealing properly with the file; additionally, the application should then close the dialog box containing the File Selector.

The above list describes the basics of File Selector support. Different File Selectors (such as SaveAs dialogs that allow the user to type in a file name) can easily be defined by adding other gadgetry to the controller. All other actions possible are internal to the file selector (e.g. the "Go To Document" button is a feature of the File Selector which is invisible to the application).

14.2.3 Messages to Handle

The list above shows two different messages that your File Selector's basic configuration: The first results from the user clicking on an entry and comes from the File Selector object, and the second is sent by the "Insert" trigger when the user clicks on it. The following descriptions show what you must do when handling these messages. (Note that in many cases only one message, the File Selector's action message, will be used; this occurs when the "Insert" trigger is actually an IC_OK or similar response trigger.)

Code Display 14-2 shows a sample handler for the message sent by the File Selector. Code Display 14-3 shows a sample handler for the message sent by the Reply Bar's Insert trigger. Both these handlers are written to go with the code in Code Display 14-1, and both handle the simplest case. Where application-specific code is required, comments have been inserted.

You should not, in general, *have* to handle any more than the File Selector's notification message. You may add your own gadgetry to add other functionality, however.

The File Selector's notification message is the message you set in *GFSI_notificationMsg*. This message should have the same definition as the prototype GEN_FILE_SELECTOR_NOTIFICATION_MSG, which is defined below. You can either define it with the same parameters, or you can define it exactly on the prototype. (The latter is recommended.) The prototype carries with the notification a word of **GenFileSelectorEntryFlags**; this record contains the following flag fields:

GFSEF_TYPE
This is a two-bit field (at offset GFSEF_TYPE_OFFSET) that describes the type of the selection. This may be one of GFSET_FILE (selection is a file), GFSET_SUBDIR (selection is a subdirectory), or GFSET_VOLUME (selection is a volume or disk). These are values of **GenFileSelectorEntryType**.

GFSEF_OPEN Set if the selection action was an "open" (i.e. double-click).

GFSEF_NO_ENTRIES
Set if no entries are in the list.

GFSEF_ERROR
Set if there was an error opening the selected entry on a double-click or a MSG_GEN_FILE_SELECTOR_OPEN_ENTRY.

GFSEF_TEMPLATE
Set if the file is a template (i.e. has GFHF_TEMPLATE set).

GFSEF_SHARED_MULTIPLE
Set if the file is shared with multiple writers (i.e. has GFHF_SHARED_MULTIPLE).

GFSEF_SHARED_SINGLE
Set if the file is shared and has a single writer (i.e. has GFHF_SHARED_SINGLE).

GFSEF_READ_ONLY
Set if the file is read-only (i.e. has FA_RDONLY).

GFSEF_PARENT_DIR
Set if the current selection is the parent directory (i.e. the first entry in the file selector's list).

■ GEN_FILE_SELECTOR_NOTIFICATION_MSG

```
void    GEN_FILE_SELECTOR_NOTIFICATION_MSG(
        word                        entryNum,
        GenFileSelectorEntryFlags  entryFlags);
```

This prototype should be used for all notification messages sent out by the File Selector; that is, any message you set in *GFSI_notificationMsg* should be based on this prototype.

14.2

Source: The GenFileSelector object.

Destination: The object specified in *GFSI_destination*.

Parameters: *entryNum* The entry number selected in the list.

entryFlags A **GenFileSelectorEntryFlags** record describing the selection. To get the entry type from the flags record, use the macro GFS_GET_ENTRY_TYPE, described after this reference entry.

Return: Nothing.

Structures: **GenFileSelectorEntryFlags** is a record with the following flags:

```
typedef WordFlags GenFileSelectorEntryFlags;
#define GFSEF_TYPE                  0xc000
#define GFSEF_OPEN                  0x2000
#define GFSEF_NO_ENTRIES            0x1000
#define GFSEF_ERROR                 0x0800
#define GFSEF_TEMPLATE              0x0400
#define GFSEF_SHARED_MULTIPLE       0x0200
#define GFSEF_SHARED_SINGLE         0x0100
#define GFSEF_READ_ONLY             0x0080
#define GFSEF_PARENT_DIR            0x0040
#define GFSEF_TYPE_OFFSET     14
```

The GFSEF_TYPE field is two bits defined by one of the following **GenFileSelectorEntryType** constants. These two bits define the type of entry selected.

```
typedef ByteEnum GenFileSelectorEntryType;
#define GFSET_FILE      0
#define GFSET_SUBDIR    1
#define GFSET_VOLUME    2
```

◆Objects

Interception: The notification message must be intercepted and handled for the selection of a file to have any effect in your application. There is no default behavior, except in the document control objects (which create their own File Selectors).

■ GFS_GET_ENTRY_TYPE

```
byte     GFS_GET_ENTRY_TYPE(flags)
         word   flags;
```

This macro extracts the high two bits from a **GenFileSelectorEntryFlags** record delivered as a parameter of a File Selector's notification message. Compare the value to GFSET_FILE, GFSET_SUBDIR, and GFSET_VOLUME.

Code Display 14-2 Handling a File Selector Selection

```
/*
 * C handler for MSG_MY_APP_FILE_SELECTED. The message is sent by the File Selector
 * object (MyFileSel) to the application's Process object (MyProcessClass) when the
 * user clicks on an entry in the displayed file list. This code display is per the
 * setup in Code Display 14-1.
 */

/*
 * Format of the message:
 * void MSG_MY_APP_FILE_SELECTED(word entryNum,
 *                                       GenFileSelectorEntryFlags entryFlags)
 */

@method MyProcessClass, MSG_MY_APP_FILE_SELECTED {
    /* First check if this is an OPEN operation (double-click). If so,
     * then test whether the selection is a file. If so, simulate the "Insert"
     * trigger by sending the trigger's message to ourselves. If it is not a file,
     * or if the operation is not OPEN, we need do nothing. */

    if (GFS_GET_ENTRY_TYPE(entryFlags) & GFSEF_OPEN) {    /* Is the operation
                                                       * a double-click? */
        if (GFS_GET_ENTRY_FLAGS(entryFlags) & GFSET_FILE) {
                                                 /* Is the selection a file? */
            /* Execute application-specific code here. */
        }
    }
}
```

```
/* Note that we do not necessarily have to handle double-click operations in this
 * way. Since a double-click automatically activates the GenInteraction’s default
 * element (typically an “Ok” trigger), we can simply handle the press of the
 * “Ok” trigger as shown in Code Display 14-3. */
```

Code Display 14-3 Handling a File Selector “OK” Button

```
/* C handler for MSG_MY_APP_INSERT_TRIGGER_SELECTED, the message sent by the Insert
 * trigger to the application’s Process object when the user clicks on it.
 * Format of this message: void MSG_MY_APP_INSERT_TRIGGER_SELECTED().
 * This code display is per the setup shown in Code Display 14-1. */

@method MyProcessClass, MSG_MY_APP_INSERT_TRIGGER_SELECTED {
    /* Declare a local dword of flags and entry number. */
    dword selFlags;

    /* First, retrieve the selection number and flags from the File Selector.
     * For now, we can ignore the selection name and path. To retrieve this
     * information, send the message MSG_GEN_FILE_SELECTOR_GET_SELECTION to the
     * File Selector. This message will return a dword (selFlags) that contains the
     * selection’s entry number and a GenFileSelectorEntryFlags record. (To ignore
     * the selection name, pass a null pointer with the message.) */

    selFlags = @call MyFileSel::MSG_GEN_FILE_SELECTOR_GET_SELECTION(NULL);

    /* Next, determine whether the selection is a file. To do this, check the
     * returned GenFileSelectorEntryFlags record against GFSET_FILE. If the result
     * is true, the selection is a file. If false, it is a directory or volume. If
     * the selection is a file, we will operate on it appropriately (this is
     * specific to each application). If not, we will direct the File Selector to
     * open the entry. Note that we can assume an OPEN operation is in progress
     * because the only two ways to get to this point are through a double-click
     * on a selection and through a click on the “Insert” trigger.
     */

    /* Note the use of the two macros GFS_GET_ENTRY_FLAGS and GFS_GET_ENTRY_NUMBER.
     * The first extracts the GenFileSelectorEntryFlags record from the dword
     * selFlags, and the second extracts the entry number from selFlags. */

    if (GFS_GET_ENTRY_FLAGS(selFlags) == GFSET_FILE) {/* the selection is a file */

        /* Execute application-specific code here. */

    } else {                    /* the selection is a volume or directory */
```

```
        /* Now direct the File Selector to open the entry. Use the message
         * MSG_GEN_FILE_SELECTOR_OPEN_ENTRY, which returns a flag; if the flag is
         * set, then an error occurred. If the flag is clear, the operation
         * succeeded. */

        if (@call MyFileSel::MSG_GEN_FILE_SELECTOR_OPEN_ENTRY(
                                        GFS_GET_ENTRY_NUMBER(selFlags))) {
            /* Provide a proper error message or beep sound. */
        }
}
```

14.2.4 Some Common Customizations

Although the basics are all covered under the previous sections and you need nothing more to add a File Selector to your application, you will probably want to customize it somewhat. Typical simple customizations include limiting the types of files displayed, restricting the search to a single directory or volume, limiting the display to directories only (no files), and limiting the search to only those files with a given filename extension.

Each of these is shown in the displays following this section. Notice that the only changes you need to make to implement any of these customizations is to the File Selector's Goc code. You will not need to handle any other messages or add other code to your application. These customizations are shown in Code Display 14-4 through Code Display 14-8 and show how to do the following (you may apply any combination):

- Show directories only.
  The File Selector will display no files, only directories.
- Show all files in a single directory.
  The File Selector will show all files but will restrict directory navigation; the user can only select files in the current directory.
- Show hidden files as well as normal files.
- Show all files created by a given geode.
  The File Selector will restrict displayed files to only those that were created by the geode with the given **tokenChars** and **tokenID**.

- Show all files with a given filename extension.
  The File Selector will restrict the files displayed to only those with the given filename extension.

These are not the only customizations you can make. Although they will satisfy the needs of the large majority of applications, you can change the File Selector's searches any way you want (you can even set up a callback routine to filter every single file or directory). The remainder of this chapter explains in detail the different things you can do to the File Selector.

Code Display 14-4 Display Only Directories

```
@object GenFileSelectorClass DirectorySelector = {
    GFSI_destination = process;
    GFSI_notificationMsg = MSG_MY_APP_DIRECTORY_SELECTED;
    GFSI_fileCriteria = FSFC_DIRS;
};
```

Code Display 14-5 Limit the Search to a Single Directory

```
@object GenFileSelectorClass SingleDirFileSelector = {
    GFSI_destination = process;
    GFSI_notificationMsg = MSG_MY_APP_FILE_SELECTED;
    GFSI_attrs =        FSA_HAS_FILE_LIST;
    GFSI_fileCriteria = FSFC_NON_GEOS_FILES | FSFC_GEOS_EXECUTABLES
                        | FSFC_GEOS_NON_EXECUTABLES;
    ATTR_GEN_PATH_DATA = {0, DIRECTORY};
                /* Limit the display to this standard directory.
                 * Note that paths may be given; relative paths are taken as
                 * relative to the specified standard path. This attribute
                 * is defined in GenClass. */
};
```

Code Display 14-6 Show Hidden Files As Well As Normal Files

```
@object GenFileSelectorClass HiddenFileSelector = {
    GFSI_destination = process;
    GFSI_notificationMsg = MSG_MY_APP_FILE_SELECTED;
        /* Specify that FA_SYSTEM files are the only ones not to be displayed. */
    ATTR_GEN_FILE_SELECTOR_FILE_ATTR = { 0, FA_SYSTEM};
};
```

Code Display 14-7 Show Only Files Created by a Given Geode

```
@object GenFileSelectorClass CreatorFileSelector = {
    GFSI_destination = process;
    GFSI_notificationMsg = MSG_MY_APP_FILE_SELECTED;
    ATTR_GEN_FILE_SELECTOR_CREATOR_MATCH = {{“CRTR”}, 0};
};
```

Code Display 14-8 Show Only Files with a Given Filename Extension

```
@object GenFileSelectorClass ExtensionFileSelector = {
    GFSI_destination = process;
    GFSI_notificationMsg = MSG_MY_APP_FILE_SELECTED;
    ATTR_GEN_FILE_SELECTOR_NAME_MASK = {“*.BAT”};/* Show files ending with BAT. */
        /* Note that the mask is case sensitive. This means that the mask must be
         * in upper case to match DOS files. */
};
```

14.3 File Selector Instance Data

The File Selector is diverse and flexible, and you can customize it in many different ways. Typically, you will only need to add or modify a few lines of Goc code to achieve the results you need; however, if you have changing needs, you can change the File Selector during execution by sending it various messages.

The File Selector has a number of attributes and variable data fields that determine its functionality. These are listed in Code Display 14-9, along with comments and the defaults that are set.

Code Display 14-9 File Selector Attributes

14.3

```
/* Following are several type definitions used in the instance data. */

typedef FileLongName    GenFileSelectorSelection;
typedef WordFlags       FileSelectorFileCriteria;
typedef WordFlags       FileSelectorAttrs;
typedef FileLongName    GenFileSelectorMask;

typedef struct {
    FileAttrs   GFSFA_match;
    FileAttrs   GFSFA_mismatch;
} GenFileSelectorFileAttrs;

typedef struct {
    GeodeAttrs  GFSGA_match;
    GeodeAttrs  GFSGA_mismatch;
} GenFileSelectorGeodeAttrs;

        /* GFSI_selection
         * The selection field contains the name of the current selection being
         * displayed by the File Selector. */
    @instance GenFileSelectorSelection   GFSI_selection = {0};

        /* GFSI_fileCriteria
         * The file criteria field contains flags that determine the search and
         * filter criteria used by the File Selector when scanning directories. */
    @instance FileSelectorFileCriteria   GFSI_fileCriteria =
                                                FSFC_DIRS | FSFC_NON_GEOS_FILES |
                                                FSFC_GEOS_EXECUTABLES |
                                                FSFC_GEOS_NON_EXECTUTABLES;
        /* Possible flags for GFSI_fileCriteria:
         * FSFC_DIRS                      FSFC_NON_GEOS_FILES
         * FSFC_GEOS_EXECUTABLES          FSFC_GEOS_NON_EXECUTABLES
         * FSFC_MASK_CASE_INSENSITIVE     FSFC_FILE_FILTER
         * FSFC_FILTER_IS_C               FSFC_TOKEN_NO_ID
         * FSFC_USE_MASK_FOR_DIRS */

        /* GFSI_attrs
         * The attributes field determines what features of the File Selector are
         * to be implemented. */
    @instance FileSelectorAttrs  GFSI_attrs =    FSA_ALLOW_CHANGE_DIRS |
                                                FSA_HAS_CLOSE_DIR_BUTTON |
                                                FSA_HAS_OPEN_DIR_BUTTON |
                                                FSA_HAS_DOCUMENT_BUTTON |
                                                FSA_HAS_CHANGE_DIRECTORY_LIST |
                                                FSA_HAS_CHANGE_DRIVE_LIST |
                                                FSA_HAS_FILE_LIST;
    /* Possible flags for GFSI_attrs:
     * FSA_ALLOW_CHANGE_DIRS        FSA_SHOW_FIXED_DISKS_ONLY
     * FSA_SHOW_FILES_DISABLED      FSA_HAS_CLOSE_DIR_BUTTON
     * FSA_HAS_OPEN_DIR_BUTTON      FSA_HAS_DOCUMENT_BUTTON
     * FSA_HAS_CHANGE_DIRECTORY_LIST FSA_HAS_CHANGE_DRIVE_LIST
     * FSA_HAS_FILE_LIST            FSA_USE_VIRTUAL_ROOT

    /* GFSI_destination and GFSI_notificationMsg
     * When a user selects a file, the File Selector sends the message
     * specified in GFSI_notificationMsg to the object specified in
     * GFSI_destination. */
@instance optr              GFSI_destination;
@instance Message           GFSI_notificationMsg;

    /* Token and creator matches
     * These two vardata fields filter files according to their tokens.
     * A file that has a token other than that set will not be displayed
     * by the File Selector. If you only want to match files with certain
     * token characters, you can set FSFC_TOKEN_NO_ID in GFSI_fileCriteria. */
@vardata GeodeToken         ATTR_GEN_FILE_SELECTOR_TOKEN_MATCH;
@vardata GeodeToken         ATTR_GEN_FILE_SELECTOR_CREATOR_MATCH;

    /* File attributes
     * This vardata field filters files according to the files’ native
     * attributes. If a file does not have the attributes specified in
     * GFSFA_match, or if it has an attribute specified in GFSFA_mismatch,
     * it will not be displayed. */
@vardata GenFileSelectorFileAttrs    ATTR_GEN_FILE_SELECTOR_FILE_ATTR;

    /* Geode attributes
     * This vardata field filters files according to the files’ geode
     * attributes. If a file does not have the attributes specified in
     * GFSGA_match, or if it has an attribute specified in GFSGA_mismatch,
     * it will not be displayed. */
@vardata GenFileSelectorGeodeAttrs   ATTR_GEN_FILE_SELECTOR_GEODE_ATTR;
```

```
        /* Mask
         * This vardata field filters files based on their names. The mask is
         * a text string matched against file names. Files which coincide with the
         * mask string are displayed; others are not. The mask string will also
         * be applied to volumes and directories if FSFC_USE_MASK_FOR_DIRS is
         * set in GFSI_fileCriteria. */
    @vardata GenFileSelectorMask ATTR_GEN_FILE_SELECTOR_NAME_MASK;

        /* Virtual Root
         * This vardata field defines a “virtual root” for the file selector. The
         * user will not be allowed to navigate above the “virtual root” except
         * with the Change Drives popup list, if available. If the Change Drives
         * list is used, the virtual root is afterwards ignored. */
     @vardata GenFilePath       ATTR_GEN_FILE_SELECTOR_VIRTUAL_ROOT;

        /* Scalable UI Support
         * This hint is used to support scalable UI within the File Selector based
         * on the application’s features (MSG_GEN_APPLICATION_GET_APP_FEATURES).
         * This hint takes an array of GenFileSelectorScalableUIEntry structures.
         * Typically, one element will have GFSSUIC_SET_FEATURES_IF_APP_FEATURE_ON,
         * and one element will have GFSSUIC_SET_FEATURES_IF_APP_FEATURE_OFF.
         * The structures are defined below. */
    @vardata GenFileSelectorScalableUIEntry HINT_FILE_SELECTOR_SCALABLE_UI_DATA;

        /* The structures for the above hint are given below.
         * GenFileSelectorScalableUIEntry is a structure with three fields; each
         * element in the array is one of these structures.
         * GenFileSelectorScalableUICommand is the type of the first field.
         * If the data in GFSSUIE_appFeature is true according to the type in
         * GFSSUIE_command, then the features in GFSSUIE_fsFeatures will be set
         * accordingly. */
    typedef ByteEnum GenFileSelectorScalableUICommand;
    #define GFSSUIC_SET_FEATURES_IF_APP_FEATURE_ON      0
    #define GFSSUIC_SET_FEATURES_IF_APP_FEATURE_OFF     1
    #define GFSSUIC_ADD_FEATURES_IF_APP_FEATURE_ON      2
    #define GFSSUIC_SET_FEATURES_IF_APP_LEVEL           3
    #define GFSSUIC_ADD_FEATURES_IF_APP_LEVEL           4

typedef struct {
    GenFileSelectorScalableUICommand    GFSSUIE_command;
    WordFlags                           GFSSUIE_appFeature;
    FileSelectorAttrs                   GFSSUIE_fsFeatures;
} GenFileSelectorScalableUIEntry;

        /* Number of files
         * This hint determines the number of files that are visible at once. */
    @vardata word               HINT_FILE_SELECTOR_NUMBER_OF_FILES_TO_SHOW;
        /* File List Width
         * This hint defines the width of the file list. The data is a number of
         * average-width characters, up to a maximum of 255.
    @vardata word               HINT_FILE_SELECTOR_FILE_LIST_WIDTH;
```

14.3.1 The GFSI_attrs Field

GFSI_attrs, MSG_GEN_FILE_SELECTOR_GET_ATTRS, MSG_GEN_FILE_SELECTOR_SET_ATTRS

The GenFileSelector has ten attributes in *GFSI_attrs*, shown below. They are stored as a word-sized record of type **FileSelectorAttrs**; they may be retrieved with MSG_GEN_FILE_SELECTOR_GET_ATTRS and set with MSG_GEN_FILE_SELECTOR_SET_ATTRS.

FSA_ALLOW_CHANGE_DIRS
This flag allows the user to change directories by double clicking; it is set by default.

FSA_SHOW_FIXED_DISKS_ONLY
This flag does not allow removable volumes (e.g. floppy disks) to be shown in the File Selector volume list. By default, this flag is off.

FSA_SHOW_FILES_DISABLED
This flag instructs the File Selector to display files as disabled. This is useful for SaveAs operations, for example, when files must be displayed but should not be selectable by the user. By default, this flag is off. Setting this flag will turn all other search criteria off, so all filenames will be displayed.

FSA_HAS_CLOSE_DIR_BUTTON
This flag causes the File Selector to include a “Close Directory” button.

FSA_HAS_OPEN_DIR_BUTTON
This flag causes the File Selector to include an “Open Directory” button.

FSA_HAS_DOCUMENT_BUTTON
This flag causes the File Selector to include a “Go to Document” button that displays the DOCUMENT directory.

FSA_HAS_CHANGE_DIRECTORY_LIST
This flag causes the File Selector to include a list that allows the user to change directories.

FSA_HAS_CHANGE_DRIVE_LIST
This flag causes the File Selector to include a list that allows the user to change drives.

FSA_HAS_FILE_LIST
This flag causes the File Selector to include a file list.

FSA_USE_VIRTUAL_ROOT
This flag causes the File Selector to use the information in the ATTR_GEN_FILE_SELECTOR_VIRTUAL_ROOT attribute. This allows changing the File Selector without altering the vardata.

■ MSG_GEN_FILE_SELECTOR_SET_ATTRS

```
void      MSG_GEN_FILE_SELECTOR_SET_ATTRS(
          FileSelectorAttrs attributes);
```

This message sets the File Selector’s *GFSI_attrs* record to a new set of attributes. The File Selector will rescan the current directory using the new attributes. If the File Selector is not visible when it receives this message, it will set the attributes but will not rescan the directory.

Source: Unrestricted.

Destination: Any GenFileSelector object.

Parameters: *attributes* A record of **FileSelectorAttrs** to set into the File Selector’s *GFSI_attrs* record.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_GEN_FILE_SELECTOR_GET_ATTRS

```
FileSelectorAttrs MSG_GEN_FILE_SELECTOR_GET_ATTRS();
```

This message returns the current attributes set in the File Selector’s *GFSI_attrs* instance data.

Source: Unrestricted.

Destination: Any GenFileSelector object.

Parameters: None.

Return: A record of **FileSelectorAttrs** representing the flags set in the File Selector's *GFSI_attrs* field.

Interception: Generally not intercepted.

14.3.2 The GFSI_fileCriteria Field

```
GFSI_fileCriteria,
MSG_GEN_FILE_SELECTOR_GET_FILE_CRITERIA,
MSG_GEN_FILE_SELECTOR_SET_FILE_CRITERIA
```

Each File Selector can limit or extend its searches in several different ways. The File Selector's instance data contains information about which types of files to include in the display, which not to include, and other special criteria. (Note that if the proper *GFSI_attrs* are set, directories and volumes will be unaffected by these scan limitations.)

The attribute field *GFSI_fileCriteria* is a record that indicates which of several search limiters should be considered in a directory scan. Note that some limitations may be invoked without a corresponding *GFSI_fileCriteria* entry. There are several possible entries in *GFSI_fileCriteria*, and any or all may be turned on at once. Each flag in the field is shown below, along with the field that it affects:

FSFC_DIRS This flag indicates that subdirectories should be displayed in the File Selector's directory scans. This flag is set by default.

FSFC_NON_GEOS_FILES
This flag indicates that non-GEOS files should be displayed. This flag is set by default.

FSFC_GEOS_EXECUTABLES
This flag indicates that GEOS executable files should be displayed. This flag is set by default.

FSFC_GEOS_NON_EXECUTABLES
This flag indicates that GEOS non-executable files (documents and other types) should be displayed. This flag is set by default.

FSFC_MASK_CASE_INSENSITIVE
This flag indicates that the filename mask (if any) should be case-insensitive. The mask applies only to files unless FSFC_USE_MASK_FOR_DIRS is also set (see below); the mask is set in ATTR_GEN_FILE_SELECTOR_NAME_MASK.

FSFC_FILE_FILTER
This flag makes the File Selector use an application-defined callback routine when scanning. The callback routine is used after all other filters have been applied. The callback routine must be defined in a subclass of **GenFileSelectorClass**.

FSFC_FILTER_IS_C
This flag indicates that the callback filter routine is written in C and follows the Pascal calling convention.

FSFC_TOKEN_NO_ID
This flag indicates that when using a token to match files, the File Selector should ignore the token ID. In other words, it will only match the token characters.

FSFC_USE_MASK_FOR_DIRS
This flag indicates that if a filename mask is used, the File Selector should apply the mask to directories as well as to files. By default, the mask is applied only to file names.

You should set the proper *GFSI_fileCriteria* attributes in your application's Goc code. However, you can change the *GFSI_fileCriteria* field at run-time with MSG_GEN_FILE_SELECTOR_SET_FILE_CRITERIA. You can also get the flags with MSG_GEN_FILE_SELECTOR_GET_FILE_CRITERIA. Both of these messages are detailed below.

■ MSG_GEN_FILE_SELECTOR_SET_FILE_CRITERIA

```
void    MSG_GEN_FILE_SELECTOR_SET_FILE_CRITERIA(
        FileSelectorFileCriteria fileCriteria);
```

This message sets the File Selector's *GFSI_fileCriteria* record to a new set of flags. If the File Selector is visible when it receives this message, it will rescan the current directory with the new attributes. Otherwise, it will set the *GFSI_fileCriteria* record but will not rescan.

Source: Unrestricted.

Destination: Any GenFileSelector object.

Parameters: *fileCriteria* A record of **FileSelectorFileCriteria** to set into the File Selector's *GFSI_fileCriteria* record.

Return: Nothing.

Interception: Generally not intercepted.

See Also: MSG_GEN_FILE_SELECTOR_SET_FILE_ATTRS,
MSG_GEN_FILE_SELECTOR_SET_TOKEN,
MSG_GEN_FILE_SELECTOR_SET_CREATOR,
MSG_GEN_FILE_SELECTOR_SET_GEODE_ATTRS,
MSG_GEN_FILE_SELECTOR_SET_MASK

■ MSG_GEN_FILE_SELECTOR_GET_FILE_CRITERIA

```
FileSelectorFileCriteria MSG_GEN_FILE_SELECTOR_GET_FILE_CRITERIA();
```

This message returns the File Selector's current *GFSI_fileCriteria* record.

Source: Unrestricted.

Destination: Any GenFileSelector object.

Parameters: None.

Return: A record of **FileSelectorFileCriteria** representing the File Selector's *GFSI_fileCriteria* flags.

Interception: Generally not intercepted.

See Also: MSG_GEN_FILE_SELECTOR_SET_FILE_ATTRS,
MSG_GEN_FILE_SELECTOR_SET_TOKEN,
MSG_GEN_FILE_SELECTOR_SET_CREATOR,
MSG_GEN_FILE_SELECTOR_SET_GEODE_ATTRS,
MSG_GEN_FILE_SELECTOR_SET_MASK

14.3.3 Matching a File's Token

```
ATTR_GEN_FILE_SELECTOR_TOKEN_MATCH,
MSG_GEN_FILE_SELECTOR_SET_TOKEN,
MSG_GEN_FILE_SELECTOR_GET_TOKEN
```

Since every GEOS file has a token associated with it, you can limit directory scans to include only those files with a given token. (Note that directories and volumes are not affected by this limitation.) Alternatively, because the token is made up of two items, the token chars and the token ID, you can arrange a search based on just the token characters (you can not search on only the ID).

Both the match token ID and the match token characters are stored in the File Selector's ATTR_GEN_FILE_SELECTOR_TOKEN_MATCH instance field. This field is of type **GeodeToken**, the structure of which is shown below:

```
typedef struct {
   TokenChars     GT_chars[TOKEN_CHARS_LENGTH];
   ManufacturerID GT_manufID;
} GeodeToken;
```

The *GT_chars* field is four characters as defined in the geode's geode parameters (**.gp**) file. The *GT_manufID* field is the manufacturer ID number allotted to the particular developer who created the geode.

If the *GFSI_fileCriteria* attribute FSFC_TOKEN_NO_ID is set, only the token characters will be matched. The ID portion of the token will be ignored by the File Selector (until FSFC_TOKEN_NO_ID is turned off). See section 14.3.2 on page 957 for more information on *GFSI_fileCriteria*.

If no token information is set, no filter will be applied based on tokens. To set the token at run-time, send MSG_GEN_FILE_SELECTOR_SET_TOKEN to the File Selector. To retrieve the token match information, send MSG_GEN_FILE_SELECTOR_GET_TOKEN.

■ MSG_GEN_FILE_SELECTOR_SET_TOKEN

```
void     MSG_GEN_FILE_SELECTOR_SET_TOKEN(
         dword               tokenChars,
         ManufacturerID      manufacturerID);
```

This message sets the ATTR_GEN_FILE_SELECTOR_TOKEN_MATCH field to the two passed values. The token set with this message will be used to filter out certain files; directories and volumes are not affected by this filter.

If the File Selector is visible when it receives this message, it will rescan the current directory using the new token information. Otherwise, the File Selector will store the passed token.

Source: Unrestricted.

Destination: Any GenFileSelector object.

Parameters: *tokenChars* The token characters of the token to match files against. Use the macro SET_TOKEN_CHARS (below) to set this dword value from the four individual characters.

manufacturerID The token ID of the token to match files against.

Return: Nothing.

Interception: Generally not intercepted.

See Also: MSG_GEN_FILE_SELECTOR_SET_FILE_CRITERIA, MSG_GEN_FILE_SELECTOR_SET_CREATOR

■ MSG_GEN_FILE_SELECTOR_GET_TOKEN

```
void    MSG_GEN_FILE_SELECTOR_GET_TOKEN(
        GetTokenCreatorParams *retValue);
```

This message returns the current token match information set in the File Selector's ATTR_GEN_FILE_SELECTOR_TOKEN_MATCH field.

Source: Unrestricted.

Destination: Any GenFileSelector object.

Parameters: *retValue* A pointer to a **GetTokenCreatorParams** structure, shown below. The structure may be empty when passed.

Return: The **GetTokenCreatorParams** structure pointed to by *retValue* will contain the token in ATTR_GEN_FILE_SELECTOR_TOKEN_MATCH.

Structures: The **GetTokenCreatorParams** structure consists of two elements: a **GeodeToken** structure containing the File Selector's token characters and manufacturer ID fields, and a reserved word. Its structure is shown below:

```
typedef struct {
    GeodeToken   GTP_Token;
    word         GTP_Unused;        /* reserved */
} GetTokenCreatorParams;
```

Interception: Generally not intercepted.

See Also: MSG_GEN_FILE_SELECTOR_SET_FILE_CRITERIA, MSG_GEN_FILE_SELECTOR_GET_CREATOR

■ SET_TOKEN_CHARS

```
dword    SET_TOKEN_CHARS(ch1, ch2, ch3, ch4);
         char   ch1, ch2, ch3, ch4;
```

This macro creates a single dword containing the four given characters. Use this macro to create the *tokenChars* parameter for MSG_GEN_FILE_SELECTOR_SET_TOKEN and MSG_GEN_FILE_SELECTOR_SET_CREATOR.

14.3.4 Matching a File's Creator App

```
ATTR_GEN_FILE_SELECTOR_CREATOR_MATCH,
MSG_GEN_FILE_SELECTOR_SET_CREATOR,
MSG_GEN_FILE_SELECTOR_GET_CREATOR
```

Most applications will store their documents in a special type of file; each of these files will include the token of the creator geode. (This is a function of the Document Control objects.)

As a result, you may limit the File Selector's searches to only those files created by a given geode. Set the token to be matched in the File Selector's ATTR_GEN_FILE_SELECTOR_CREATOR_MATCH variable data instance field. This search is employed in the same manner as a token search (above), so it is also subject to the FSFC_TOKEN_NO_ID attribute of *GFSI_fileCriteria*.

The ATTR_GEN_FILE_SELECTOR_CREATOR_MATCH field is a **GeodeToken** structure that contains both the match ID and the match characters of the creator token. The **GeodeToken** structure is shown in the previous section. You can set the creator information in the File Selector's Goc code, or you can set it at run-time with MSG_GEN_FILE_SELECTOR_SET_CREATOR. To retrieve the creator token, use MSG_GEN_FILE_SELECTOR_GET_CREATOR.

■ MSG_GEN_FILE_SELECTOR_SET_CREATOR

```
void     MSG_GEN_FILE_SELECTOR_SET_CREATOR(
         dword              tokenChars,
         ManufacturerID     manufacturerID);
```

This message sets the ATTR_GEN_FILE_SELECTOR_CREATOR_MATCH field to the two passed values. This field is a vardata instance record of type **GeodeToken**. The token set with this message will be used to filter out certain files; directories and volumes are not affected by this filter. Only files which have their creator token the same as that of the File Selector will be displayed.

If the File Selector is visible when it receives this message, it will rescan the directory using the new token as a filter. Otherwise, the File Selector will store the passed token for later use.

Source: Unrestricted.

Destination: Any GenFileSelector object.

Parameters: *tokenChars* The token characters of the creator token to match files against. Use the macro SET_TOKEN_CHARS to set this dword value from the four individual characters. This macro is shown in the previous section.

manufacturerID The token ID of the creator token to match files against.

Return: Nothing.

Interception: Generally not intercepted.

See Also: MSG_GEN_FILE_SELECTOR_SET_FILE_CRITERIA, MSG_GEN_FILE_SELECTOR_SET_TOKEN

■ MSG_GEN_FILE_SELECTOR_GET_CREATOR

```
void        MSG_GEN_FILE_SELECTOR_GET_CREATOR(
            GetTokenCreatorParams *retValue);
```

This message returns the current creator match information set in the File Selector's ATTR_GEN_FILE_SELECTOR_CREATOR_MATCH field.

Source: Unrestricted.

Destination: Any GenFileSelector object.

Parameters: *retValue* A pointer to a **GetTokenCreatorParams** structure, shown below. The structure may be empty when passed.

Return: The **GetTokenCreatorParams** structure pointed to by *retValue* will contain the token in ATTR_GEN_FILE_SELECTOR_CREATOR_MATCH.

Structures: The **GetTokenCreatorParams** structure consists of two elements: a **GeodeToken** structure containing the File Selector's token characters and manufacturer ID fields, and a reserved word. Its structure is shown below:

```
typedef struct {
    GeodeToken   GTP_token;
    word         GTP_unused;        /* reserved */
} GetTokenCreatorParams;
```

Interception: Generally not intercepted.

See Also: MSG_GEN_FILE_SELECTOR_SET_FILE_CRITERIA, MSG_GEN_FILE_SELECTOR_GET_TOKEN

14.3.5 Matching a File's Geode Attributes

```
ATTR_GEN_FILE_SELECTOR_GEODE_ATTR,
MSG_GEN_FILE_SELECTOR_SET_GEODE_ATTRS,
MSG_GEN_FILE_SELECTOR_GET_GEODE_ATTRS
```

Every geode in GEOS has a structure of type **GeodeAttrs** stored in its file. As a result, you can limit the File Selector's searches to include only a certain set of geodes—those with the attributes set in the File Selector's ATTR_GEN_FILE_SELECTOR_GEODE_ATTR vardata instance field.

ATTR_GEN_FILE_SELECTOR_GEODE_ATTR is a structure of type **GenFileSelectorGeodeAttrs**, which consists of two parts. *GFSGA_match* represents the geode attributes an acceptable file must have on, and *GFSGA_mismatch* represents the geode attributes an acceptable file must have off. The structure's definition is shown below:

```
typedef struct {
    GeodeAttrs    GFSGA_match;
    GeodeAttrs    GFSGA_mismatch;
} GenFileSelectorGeodeAttrs;
```

To retrieve the current geode match information, send the message MSG_GEN_FILE_SELECTOR_GET_GEODE_ATTRS to the File Selector object. To set them, send MSG_GEN_FILE_SELECTOR_SET_GEODE_ATTRS. If you do not explicitly set the geode match information, the File Selector will not apply the geode match filter to its directory scans.

The possible geode attributes on which you can match or mismatch are shown below. They are all flags of the **GeodeAttrs** record, a system type that is not specific to **GenFileSelectorClass**. Note that none of these flags is set by default. (Geode attributes are described in detail in “Applications and Geodes,” Chapter 6 of the Concepts Book.)

GA_PROCESS This flag indicates the geode is a process (has an initial thread).

GA_LIBRARY This flag indicates the geode is a library (exports routines).

GA_DRIVER This flag indicates the geode is a driver (has a Driver Table).

GA_KEEP_FILE_OPEN
This flag indicates the **.geo** file must stay open (e.g. is a resource that is designated as discardable).

GA_SYSTEM This flag indicates the geode is a GEOS system geode.

GA_MULTI_LAUNCHABLE
This flag indicates the geode may be loaded more than once.

GA_APPLICATION
This flag indicates the geode is a user-launchable application.

GA_DRIVER_INITIALIZED
Set on the fly by GEOS if the geode’s driver aspect has been initialized (should not be matched).

GA_LIBRARY_INITIALIZED
Set on the fly by GEOS if the geode’s library aspect has been initialized (should not be matched).

GA_GEODE_INITIALIZED
Set on the fly by GEOS if the geode’s process aspect has been initialized (should not be matched).

GA_USES_COPROC
This flag indicates that the geode uses a coprocessor if one is available.

GA_REQUIRES_COPROC
This flag indicates the geode requires a coprocessor or coprocessor emulator.

GA_HAS_GENERAL_CONSUMER_MODE
This flag indicates the geode can be run in GCM mode.

GA_ENTRY_POINTS_IN_C
This flag indicates that the geode has library/driver entry points in C.

■ MSG_GEN_FILE_SELECTOR_SET_GEODE_ATTRS

```
void      MSG_GEN_FILE_SELECTOR_SET_GEODE_ATTRS(
          GeodeAttrs          matchGeodeAttrs,
          GeodeAttrs          mismatchGeodeAttrs);
```

This message sets the File Selector's vardata instance field ATTR_GEN_FILE_SELECTOR_GEODE_ATTR to the two passed values.

If the File Selector is visible when it receives this message, it will automatically rescan the current directory, applying the new geode filter. Otherwise, the new geode match/mismatch records will be stored but no rescan will occur.

Source: Unrestricted.

Destination: Any GenFileSelector object.

Parameters: *matchGeodeAttrs* A **GeodeAttrs** record indicating the flags that every acceptable file must have set. This record will be set into *GFSGA_match*.

mismatchGeodeAttrs
A **GeodeAttrs** record indicating the flags that every acceptable file must have clear. This record will be set into *GFSGA_mismatch*.

Return: Nothing.

Interception: Generally not intercepted.

See Also: MSG_GEN_FILE_SELECTOR_SET_FILE_CRITERIA

■ MSG_GEN_FILE_SELECTOR_GET_GEODE_ATTRS

```
dword      MSG_GEN_FILE_SELECTOR_GET_GEODE_ATTRS();
```

This message returns the File Selector's current geode attribute match information.

Source: Unrestricted.

Destination: Any GenFileSelector object.

Parameters: None.

Return: The returned dword consists of the two **GeodeAttrs** records stored in the File Selector's ATTR_GEN_FILE_SELECTOR_GEODE_ATTR vardata field. To extract the match attributes (*GFSGA_match*), use the macro GET_MATCH_ATTRS. To extract the mismatch attributes (*GFSGA_mismatch*), use GET_MISMATCH_ATTRS.

Interception: Generally not intercepted.

See Also: MSG_GEN_FILE_SELECTOR_SET_FILE_CRITERIA

■ GET_MATCH_ATTRS

```
word    GET_MATCH_ATTRS(attr);
        dword  attr;
```

This macro extracts the *GFSGA_match* record (**GeodeAttrs**) from the given dword (returned by MSG_GEN_FILE_SELECTOR_GET_GEODE_ATTRS).

■ GET_MISMATCH_ATTRS

```
word    GET_MISMATCH_ATTRS(attr);
        dword  attr;
```

This macro extracts the *GFSGA_match* record (**GeodeAttrs**) from the given dword (returned by MSG_GEN_FILE_SELECTOR_GET_GEODE_ATTRS).

14.3.6 Masking File Names

ATTR_GEN_FILE_SELECTOR_NAME_MASK, MSG_GEN_FILE_SELECTOR_SET_MASK, MSG_GEN_FILE_SELECTOR_GET_MASK

By setting up a filename mask, you can limit the File Selector's displays to only those files that conform to the mask characters. For example, to show only those files with the .BAT extender, you could set the mask to *.BAT.

Only one mask may be in use at any given time. If you need to mask for two different sets of characters (e.g. *.BAT and *.EXE), you will need to subclass the File Selector and modify its behavior. (See section 14.3.8 on page 972 for more information.)

The mask is stored in the File Selector's variable data instance attribute ATTR_GEN_FILE_SELECTOR_NAME_MASK and consists of a null-terminated character string. This field is defined as type **GenFileSelectorMask**, the same as **FileLongName**. This string may contain the DOS "*****" and "**?**" wildcard characters. The filter applied by the File Selector is the same as that applied by the **FileEnum()** routine detailed in "FileEnum()" on page 655 of "File System," Chapter 17 of the Concepts Book. In short, the mask is taken as a character string matched to the entire file name (thus, a mask of ***.*** implies all files with a period in their names; the period separating a filename and an extension will not count as a period).

Normally, the mask is applied only to files and not to directories or volumes. However, if the *GFSI_fileCriteria* attribute FSFC_USE_MASK_FOR_DIRS is set, directories will also be subject to the mask. (Volumes are never subject to the mask filter.)

By default, the mask filter is case-sensitive. You can make the mask filter be case-insensitive, however, by setting FSFC_MASK_CASE_INSENSITIVE in the File Selector's *GFSI_fileCriteria* instance field.

You can set the mask in your application's Goc code or by sending the message MSG_GEN_FILE_SELECTOR_SET_MASK to the File Selector. To retrieve the current mask, use MSG_GEN_FILE_SELECTOR_GET_MASK. If you do not explicitly set a mask string, the File Selector will not make any filename mask comparisons.

■ MSG_GEN_FILE_SELECTOR_SET_MASK

```
void      MSG_GEN_FILE_SELECTOR_SET_MASK(
          char   *mask);
```

This message sets the ATTR_GEN_FILE_SELECTOR_NAME_MASK variable instance field of the File Selector. This field contains a null-terminated character string (of type **GenFileSelectorMask**); during a directory scan, the File Selector checks all filenames against this mask, and only those files that contain the mask characters are displayed. The mask string may contain DOS wildcard characters ("*****" and "**?**").

The mask works for both DOS and GEOS files. However, unless the FSFC_USE_MASK_FOR_DIRS flag is set in the *GFSI_fileCriteria* attribute, directories will be unaffected by the mask. Volumes are unaffected in any case.

If the File Selector is visible on the screen when it receives this message, it will automatically rescan the current directory with the new mask. Otherwise, the new mask will be stored for later use.

Source: Unrestricted.

Destination: Any GenFileSelector object.

Parameters: *mask* A pointer to the mask string to be set. This is a null-terminated character string.

Return: Nothing.

Interception: Generally not intercepted.

See Also: MSG_GEN_FILE_SELECTOR_SET_FILE_CRITERIA

■ MSG_GEN_FILE_SELECTOR_GET_MASK

```
void    MSG_GEN_FILE_SELECTOR_GET_MASK(
        char   *mask);
```

This message returns the mask string in the File Selector's ATTR_GEN_FILE_SELECTOR_NAME_MASK field.

Source: Unrestricted.

Destination: Any GenFileSelector object.

Parameters: *mask* A pointer to an empty character string. The string must be at least **sizeof(GenFileSelectorMask)** characters.

Return: The character string pointed to by *mask* will contain the mask string set in ATTR_GEN_FILE_SELECTOR_NAME_MASK.

Interception: Generally not intercepted.

See Also: MSG_GEN_FILE_SELECTOR_SET_MASK, MSG_GEN_FILE_SELECTOR_SET_FILE_CRITERIA

14.3.7 Matching a File's File Attributes

```
ATTR_GEN_FILE_SELECTOR_FILE_ATTR,
MSG_GEN_FILE_SELECTOR_GET_FILE_ATTRS,
MSG_GEN_FILE_SELECTOR_SET_FILE_ATTRS
```

All files have a number of attributes stored in a record of type **FileAttrs**. (For full information on **FileAttrs**, see "File System," Chapter 17 of the Concepts Book.) To filter files based on the **FileAttrs** record, set your File Selector's variable data instance field ATTR_GEN_FILE_SELECTOR_FILE_ATTR to reflect the attributes that must be on and those that must be off.

ATTR_GEN_FILE_SELECTOR_FILE_ATTR is a structure of type **GenFileSelectorFileAttrs**, shown below. This structure contains two fields: *GFSFA_match* represents the attributes an acceptable file has on, and *GFSFA_mismatch* represents the attributes an acceptable file has off.

```
typedef struct {
    FileAttrs    GFSFA_match;
    FileAttrs    GFSFA_mismatch;
} GenFileSelectorFileAttrs;
```

The allowable file attributes are listed below and are discussed fully in “File System,” Chapter 17 of the Concepts Book. If you don’t explicitly set the file attribute limitations, the File Selector will automatically filter out all files with either FA_SYSTEM or FA_HIDDEN.

FA_ARCHIVE This flag indicates that the file requires a backup.

FA_SYSTEM This flag indicates that the file used by DOS.

FA_HIDDEN This flag indicates that the file not seen by normal searches.

FA_RDONLY This flag indicates that the file is read-only.

Note: For directories not to be filtered out when FSFC_DIRS isn’t set, you must mismatch FA_SYSTEM and FA_HIDDEN. That is, to show only those subdirectories which match your filter criteria, ensure FSFC_DIRS is not set, and set mismatch attributes thus:

```
ATTR_GEN_FILE_SELECTOR_FILE_ATTR = {
                        0,
                        FA_HIDDEN | FA_SYSTEM
}
```

You may retrieve the current file attributes by sending the message MSG_GEN_FILE_SELECTOR_GET_FILE_ATTRS to the File Selector object. You may set new file attribute limitations by sending it the message MSG_GEN_FILE_SELECTOR_SET_FILE_ATTRS.

■ MSG_GEN_FILE_SELECTOR_SET_FILE_ATTRS

```
void      MSG_GEN_FILE_SELECTOR_SET_FILE_ATTRS(
          FileAttrs setAttrs,
          FileAttrs clearAttrs);
```

This message sets the ATTR_GEN_FILE_SELECTOR_FILE_ATTR vardata instance field of the File Selector. If the File Selector is visible on the screen when it receives this message, it will rescan the directory immediately with the new attributes.

Source: Unrestricted.

Destination: Any GenFileSelector object

Parameters: *setAttrs* A **FileAttrs** record to be set into *GFSFA_match*. Only files with these flags set will pass the filter.

clearAttrs A **FileAttrs** record to be set into *GFSFA_mismatch*. Only files with these flags cleared will pass the filter.

Return: Nothing.

Interception: Generally not intercepted.

See Also: MSG_GEN_FILE_SELECTOR_SET_FILE_CRITERIA

■ MSG_GEN_FILE_SELECTOR_GET_FILE_ATTRS

```
word      MSG_GEN_FILE_SELECTOR_GET_FILE_ATTRS();
```

This message returns the ATTR_GEN_FILE_SELECTOR_FILE_ATTR of the File Selector. This attribute contains two byte-sized fields which represent the match and mismatch attributes (see above).

Source: Unrestricted.

Destination: Any GenFileSelector object.

Parameters: None.

Return: A word value: The high byte represents the mismatch attributes, and the low byte represents the match attributes. To extract these two fields from the returned value, use the macros GET_MATCH_FILE_ATTRS and GET_MISMATCH_FILE_ATTRS (below).

Interception: Generally not intercepted.

See Also: MSG_GEN_FILE_SELECTOR_SET_FILE_CRITERIA

GET_MATCH_FILE_ATTRS

```
byte    GET_MATCH_FILE_ATTRS(attr);
        word    attr;
```

This macro extracts the *GFSFA_match* portion of the word value returned by MSG_GEN_FILE_SELECTOR_GET_FILE_ATTRS. The extracted value is a record of **FileAttrs**.

14.3

GET_MISMATCH_FILE_ATTRS

```
byte    GET_MISMATCH_FILE_ATTRS(attr);
        word    attr;
```

This macro extracts the *GFSFA_mismatch* portion of the word value returned by MSG_GEN_FILE_SELECTOR_GET_FILE_ATTRS. The extracted value is a record of **FileAttrs**.

14.3.8 Searching Via Callback Routine

MSG_GEN_FILE_SELECTOR_GET_FILTER_ROUTINE, GenFileSelectorFilterRoutine

If the built-in search limitations do not provide the functionality you require, you can set up a callback routine that the File Selector will call for each file or subdirectory in a directory scan. This routine will serve as the final filter for each file; if a file passes all other filters, it will be subjected to your routine. This procedure adds overhead to any scan, of course, so you should try to use the built-in File Selector functions whenever possible.

To set the File Selector to use the callback functionality, you must set FSFC_FILTER_FILE in the *GFSI_fileCriteria* field. If your callback routine is written in C, you must also set FSFC_FILTER_IS_C; the routine must also follow the Pascal calling convention.

In order to take advantage of the callback functionality, you must first subclass **GenFileSelectorClass** to handle the message MSG_GEN_FILE_SELECTOR_GET_FILTER_ROUTINE, which the File Selector will send to itself before beginning the directory scan. Your handler should return the address of the callback routine along with a pointer to an array of attributes that will be passed to it.

Objects

The callback routine can assume that if it is being called, the file has passed all other filters—the callback is therefore the last word on acceptance or rejection of the file. In essence, the callback is asked whether the file should be rejected: It should return the constant TRUE if the file is to be rejected but FALSE if it is not to be rejected.

As mentioned above, you can designate an array of extended file attributes that will be passed to the callback routine. This array contains a number of **FileExtAttrDesc** structures, each of which defines a certain attribute and some extra data for the attribute. This structure is shown below:

```
typedef struct {
    FileExtendedAttribute FEAD_attr;
                        /* file attribute to
                         * get or set */
    void    * FEAD_value;
                        /* pointer to a buffer
                         * or new value */
    word    FEAD_size;  /* size of buffer or
                         * new value */
    char    * FEAD_name;
                        /* if custom attribute,
                         * pointer to name */
} FileExtAttrDesc;
```

In the above context, *FEAD_value* is meaningless and *FEAD_size* is the number of bytes required to hold the value of the attribute. Note that this array is not passed to the callback routine; instead, it specifies the structure and composition of the values that will be.

An example of filtering files using a callback routine is shown in Code Display 14-10. The **FileExtAttrDesc** structure is defined and detailed in “File System,” Chapter 17 of the Concepts Book.

Code Display 14-10 Filtering Files Via a Callback Routine

```
/*******************************************************************************
 * This is a sample handler for MSG_GEN_FILE_SELECTOR_GET_FILTER_ROUTINE.
 * You should be able to use this code in your subclass. Note that the routine
 * name “FilterFileSelectorRoutine” is application-specific; it can be whatever
 * you want. It must be declared as static Boolean, however. See below.
 * PARAMETERS:
 * void (GenFileSelectorGetFilterRoutineResults *filter)
 *******************************************************************************/

@method FilterFileSelectorClass, MSG_GEN_FILE_SELECTOR_GET_FILTER_ROUTINE {

    /* We need to return a virtual pointer to the filter routine so the
     * file selector can lock and unlock the routine’s code resource as
     * appropriate. Unfortunately, the compiler optimizes the simple
     * implementation (just assigning filter->filterRoutine the address of
     * the routine) by storing CS for the segment, which is unhelpful. So
     * we need a static variable holding the routine’s address instead. */

    static GenFileSelectorFilterRoutine *const filterRoutine =
                                         &FilterFileSelectorRoutine;

    /* Specify the address of the routine to call. It need not be locked into
     * memory, as the GenFileSelector will do that for us. */

    filter->filterRoutine = filterRoutine;

    /* Specify the additional attributes we need to look at in our filter
     * routine. As with the filterRoutine, these need not be in fixed or
     * locked memory. */

    filter->filterAttrs = &filterFileSelectorFilterAttrs;
}

/*******************************************************************************
 * This is a sample File Selector callback routine. It may be called whatever
 * you want and must be declared static Boolean.
 * This routine is called once per file or directory that passes all other File
 * Selector filters. This routine is limited to examining the attributes of the
 * file. It may not do anything that could cause the File Selector’s instance chunk
 * to move in memory.
 * Return: TRUE to reject the file, FALSE to accept the file.
 *******************************************************************************/

static Boolean FilterFileSelectorRoutine(optr oself,
                                FileEnumCallbackData *fecd, word frame)

/* The code of the routine is not included here; to see it, look at the FSFilter
 * sample application. */
```

Objects

```
        /* Following is the array of attributes examined by our filter routine.
         * This array contains an arbitrary number of elements, the last of which
         * has FEA_END_OF_LIST as its FEAD_attr field. FEAD_value is unused in this
         * context. FEAD_size is set to the total number of bytes needed for each
         * attribute.
         * Each structure's fields are FEAD_attr, FEAD_value, FEAD_size,
         * and FEAD_name */

const FileExtAttrDesc filterFileSelectorFilterAttrs[] = {
    { FEA_NAME, 0, sizeof(FileLongName), NULL },
    { FEA_FILE_ATTR, 0, sizeof(FileAttrs), NULL },
    { FEA_FILE_TYPE, 0, sizeof(GeosFileType), NULL },
    { FEA_END_OF_LIST, 0, 0, NULL }
};
```

■ MSG_GEN_FILE_SELECTOR_GET_FILTER_ROUTINE

```
void    MSG_GEN_FILE_SELECTOR_GET_FILTER_ROUTINE(
        GenFileSelectorGetFilterRoutineResults *filter);
```

This message returns the address of the callback routine used by the File Selector as well as the array of **FileExtAttrDesc** structures specifying the callback's parameters.

For the callback function to be used, you must set FSFC_FILE_FILTER in the File Selector's *GFSI_fileCriteria* instance data field. If your callback routine is in C, you must also set FSFC_FILTER_IS_C. In addition, the callback routine must follow the Pascal calling convention.

Source: Sent by the File Selector object to itself before calling **FileEnum()**. Also may be used by other objects to retrieve the callback information for the File Selector.

Destination: Any GenFileSelector object.

Parameters: *filter* A pointer to an empty return value structure. This structure is detailed below.

Return: The structure pointed to by *filter* must be filled by this method.

Structures: The single parameter is a pointer to an empty structure of type **GenFileSelectorGetFilterRoutineResults**. This structure has two parts: *filterRoutine* is a pointer to your callback routine, and *filterAttrs* is a pointer to an array of **FileExtAttrDesc** structures. You must fill in this structure in your handler so the File Selector knows how to call your callback routine. The structure is shown below:

```
typedef struct {
    GenFileSelectorFilterRoutine    *filterRoutine;
    const FileExtAttrDesc           *filterAttrs;
} GenFileSelectorGetFilterRoutineResults;
```

Interception: If a callback routine is to be used, you must subclass and handle this message to return the proper structures.

Tips: If your callback has no requirements other than the file name, you can pass NULL in the *filterAttrs* field of *filter*.

Warnings: The handler for this message must return a virtual segment for the callback routine in *filter*'s *filterRoutine* field. This can cause problems with most compilers; to avoid these problems, set a static variable in your method as shown in Code Display 14-10, below (taken from the FSFilter sample application).

See Also: FileEnum()

■ GenFileSelectorFilterRoutine()

```
Boolean GenFileSelectorFilterRoutine(
        optr                     oself,
        FileEnumCallbackData     *fecd,
        word                     frame);
```

This routine is defined by your application and is called for each file in a directory scan. It may be of any name of your choice as shown in Code Display 14-10. It serves as the final filter; when called, it can assume the subject file has passed all other filters applied. This routine is set up and executed in the same manner as the callback routine for **FileEnum()**; see “FileEnum()” on page 655 of “File System,” Chapter 17 of the Concepts Book for complete details.

Parameters:

oself	The optr of the File Selector object
fecd	A pointer to a **FileEnumCallbackData** structure.
frame	An inherited stack frame passed by **FileEnum()** to each of its helper routines.

Return: Your callback must return the constant FALSE if the file should be accepted, TRUE if the file should be rejected.

◆ Objects

Warnings: Your routine must not do anything that may cause the File Selector or its object block to move in memory (adding variable data fields, for instance).

See Also: FileEnum()

14.3.9 Resetting a Filter

Once a filter is set (such as a filename mask or a file attribute match/mismatch filter), you may wish to delete the filter. To remove a filter designated by a flag in an instance field, simply clear the flag with the appropriate message.

If the filter is applied via a variable data field, however, you will have to use MSG_META_DELETE_VARDATA. This message is detailed in **MetaClass** and must be passed the name of the variable data field to be removed. Removal of the field equates to removal of the filter.

14.4 File Selector Use

Besides setting the File Selector's attributes and dynamically manipulating its instance data, you can change its display and current file lists using several messages. You can also retrieve its current state with other messages. These functions are detailed in the following sections.

14.4.1 When a User Selects a File

```
GFSI_destination, GFSI_notificationMsg,
MSG_GEN_FILE_SELECTOR_GET_ACTION,
MSG_GEN_FILE_SELECTOR_SET_ACTION
```

When a user selects an entry in the File Selector's displayed list, either by double-clicking or by clicking on an "Open" button, the File Selector will notify a predetermined object or process that a selection has been made. In addition, if the selection is a volume or a directory, the File Selector will automatically (on a double-click) open the selection and show the new file list.

When defining your GenFileSelector, you must designate an output object and a message that will be sent to it when the user makes a selection. The message is stored in the *GFSI_notificationMsg* instance field, and the object's optr is stored in the *GFSI_destination* instance field. Note that instead of an optr, you may use a **TravelOption** as the output destination (TO_TARGET, TO_FOCUS, etc.); you can also use ATTR_GEN_DESTINATION_CLASS to specify the class of the output object to ensure the notification message is only delivered to a class that can handle it.

When the user makes a selection, the File Selector will send the notification message to the destination object along with flags indicating whether the selection was a single or double click. The message is defined by the class that will receive it; for example, the File Selector definition in Code Display 14-1 designates MSG_MY_APP_FILE_SELECTED as the notification message that will be sent to the application's process object (the destination object).

You must set both the notification message and the destination object in your File Selector's Goc code. If you don't, no message will be sent, and your application will ignore user input to the File Selector. (The File Selector will still allow the user to navigate throughout the file system, and your application can query the File Selector as to its current selection, however.)

You may retrieve or set the File Selector's notification message and destination object at run-time by sending the following messages:

MSG_GEN_FILE_SELECTOR_GET_ACTION
Returns the File Selector's current notification message and destination object.

MSG_GEN_FILE_SELECTOR_SET_ACTION
Sets the File Selector's *GFSI_destination* and *GFSI_notificationMsg* fields to new values.

■ MSG_GEN_FILE_SELECTOR_GET_ACTION

```
void    MSG_GEN_FILE_SELECTOR_GET_ACTION(
        GetActionParams *retValue);
```

This message returns the File Selector's output object and notification message.

Source: Unrestricted.

Destination: Any GenFileSelector object.

Parameters: *retValue* A pointer to an empty **GetActionParams** structure (detailed below), which will be filled with the return data.

Return: The **GetActionParams** structure pointed to by *retValue* will contain the output optr and notification message set for the File Selector.

Structures: The **GetActionParams** structure is shown below:

```
typedef struct {
    Message  GAP_message; /* GFSI_notificationMsg */
    word     GAP_unused;  /* Internal field */
    optr     GAP_output;  /* GFSI_destination */
} GetActionParams;
```

Interception: Generally not intercepted.

■ MSG_GEN_FILE_SELECTOR_SET_ACTION

```
void    MSG_GEN_FILE_SELECTOR_SET_ACTION(
        optr                actionOD,
        Message             actionMessage);
```

This message sets the File Selector's *GFSI_notificationMsg* field to *actionMessage* and the *GFSI_destination* field to *actionOD*.

Source: Unrestricted.

Destination: Any GenFileSelector object.

Parameters: *actionOD* The optr of the new output object.

actionMessage The new output notification message.

Return: Nothing.

Interception: Generally not intercepted.

14.4.2 The Current Selection

```
GFSI_selection
```

Most directory and volume navigation is handled internally by the File Selector, and the application has no need to know what is happening until a file is actually selected.

The File Selector maintains the name of the current selection in its *GFSI_selection* instance data field. The current full path, volume name, and disk are stored in variable data entries defined by **GenClass**. The application can easily, at any time, retrieve or set the current selection, path, or volume. Volume names, paths, and file names are all stored as null-terminated character strings in the GEOS character set.

Current path and directory are supported by **GenClass** and are not specific to **GenFileSelectorClass**.

The File Selector's current path is stored in a variable data field called ATTR_GEN_PATH_DATA. This field is a **GenFilePath** structure, defined below. This structure stores both an absolute path and the handle of the disk on which the path resides. For the GenFileSelector, this structure represents the currently-displayed directory.

```
typedef struct {
    DiskHandle  GFP_disk;
    PathName    GFP_path;
} GenFilePath;
```

To retrieve the current path, send the File Selector a MSG_GEN_PATH_GET. To set the path, send it MSG_GEN_PATH_SET. To retrieve just the disk handle, send MSG_GEN_PATH_GET_DISK_HANDLE. Each of these messages is detailed fully in "GenClass," Chapter 2.

14.4.2.1 Traversing the File System

```
MSG_GEN_FILE_SELECTOR_UP_DIRECTORY,
MSG_GEN_FILE_SELECTOR_OPEN_ENTRY
```

To set the currently-displayed directory, you can send one of the following messages to the File Selector object (these are all, of course, subject to the File Selector's attributes and applied filters):

MSG_GEN_FILE_SELECTOR_UP_DIRECTORY
Cause the File Selector to go up one directory. If already at the root, switch to volume selection. Does nothing if FSA_ALLOW_CHANGE_DIRS is not set in *GFSI_attrs*.

MSG_GEN_FILE_SELECTOR_OPEN_ENTRY
Cause the File Selector to attempt to open the passed entry in the current directory. Nothing will be done if the passed entry is a file; this message opens only directories and volumes. Use

this with MSG_GEN_FILE_SELECTOR_GET_SELECTION when a user clicks on an "Open" button (or something similar) in the File Selector's dialog box.

■ MSG_GEN_FILE_SELECTOR_UP_DIRECTORY

```
void      MSG_GEN_FILE_SELECTOR_UP_DIRECTORY();
```

This message causes the File Selector to go up one directory in the directory tree. If the File Selector is already in the root directory, it will switch to the volume list (if FSA_ALLOW_CHANGE_VOLUMES is set in *GFSI_attrs*). This message is only valid when the File Selector is visible on the screen.

Source: Unrestricted.

Destination: Any GenFileSelector object that is on the screen.

Interception: Generally not intercepted.

■ MSG_GEN_FILE_SELECTOR_OPEN_ENTRY

```
Boolean   MSG_GEN_FILE_SELECTOR_OPEN_ENTRY(
          word   entryNumber);
```

This message causes the File Selector to open and display the contents of the entry specified. It returns an error flag: if the entry opens successfully, the error flag is returned *false*; if some error occurs, the flag is returned *true*.

The entry specified with this message must be in the current file list. First, retrieve the entry number from the File Selector with the message MSG_GEN_FILE_SELECTOR_GET_SELECTION. If the entry can be opened (i.e. it is in the current file list and it is a volume or directory), the File Selector will open it, scan the directory, and display the new file list. If the entry is not a directory or volume, the File Selector will do nothing. This message is only valid when the File Selector is visible on the screen.

Source: Unrestricted.

Destination: Any GenFileSelector object that is on the screen.

Parameters: *entryNumber* The number of the entry to be opened.

Return: An error flag: *true* if error, *false* if no error.

Interception: Generally not intercepted.

14.4.2.2 The Current Selection

```
MSG_GEN_FILE_SELECTOR_SET_SELECTION,
MSG_GEN_FILE_SELECTOR_SET_FULL_SELECTION_PATH,
MSG_GEN_FILE_SELECTOR_GET_SELECTION,
MSG_GEN_FILE_SELECTOR_GET_FULL_SELECTION_PATH
```

The File Selector keeps track of which item is currently designated as the *selection*. The selection appears on the screen as the only highlighted entry in the current list (the highlighted entry may be scrolled out of the view but remains the selection). Additionally, the selection is the file or folder on which any operations (such as open) will be executed.

The *GFSI_selection* field of the GenFileSelector's instance data is a null-terminated character string representing the name of the selection. The format of the selection string depends on the context: If the File Selector is displaying the volume list and thus the selection is a volume label, the string will consist of a drive name followed by a colon. (For non-removable disks, the colon will be followed by [*volume name*].) If the File Selector is displaying a directory and the selection is a file or directory, the selection string will consist of the name of the file or directory selected. The selection does not contain any information about the current volume or directory.

By sending the following messages, you can retrieve or set the current selection in your application. These messages affect ATTR_GEN_PATH_DATA, managed by **GenClass** for the File Selector, as well as the *GFSI_selection* instance field.

MSG_GEN_FILE_SELECTOR_SET_SELECTION
Attempt to set the selection to the given volume, file, or directory name. The given file or subdirectory must be in the currently-displayed list.

MSG_GEN_FILE_SELECTOR_SET_FULL_SELECTION_PATH
Attempt to set the path and selection to the given volume, file, or directory name. The passed selection must be in the directory at the end of the given path.

MSG_GEN_FILE_SELECTOR_GET_SELECTION
Gives a null-terminated character string representing the name of the current selection. This may be a volume, directory, or file name.

MSG_GEN_FILE_SELECTOR_GET_FULL_SELECTION_PATH
Gives the disk handle as well as a null-terminated character string representing the full path name of the current selection, excluding the drive name.

The selection retrieval messages return two word values in a single dword argument. These two values are a record of **GenFileSelectorEntryFlags**, flags that indicate the type of selection and operation underway, and an integer that indicates the place of the selection in the current file list. Two macros allow you to extract these values from the dword argument:

GFS_GET_ENTRY_NUMBER
Extracts the entry number from the given dword.

GFS_GET_ENTRY_FLAGS
Extracts the selection's flags record.

■ GFS_GET_ENTRY_NUMBER

```
word      GFS_GET_ENTRY_NUMBER(arg);
          dword  arg;
```

This takes the return value of MSG_GEN_FILE_SELECTOR_GET_SELECTION and MSG_GEN_FILE_SELECTOR_GET_FULL_SELECTION_PATH and returns the entry number of the selection.

■ GFS_GET_ENTRY_FLAGS

```
word      GFS_GET_ENTRY_FLAGS(arg);
          dword  arg;
```

This takes the return value of MSG_GEN_FILE_SELECTOR_GET_SELECTION and MSG_GEN_FILE_SELECTOR_GET_FULL_SELECTION_PATH and returns the **GenFileSelectorEntryFlags** for the selection.

■ MSG_GEN_FILE_SELECTOR_SET_SELECTION

```
Boolean   MSG_GEN_FILE_SELECTOR_SET_SELECTION(
          char   *selection);
```

This message causes the File Selector to attempt to select a given file, directory, or volume from the currently-displayed file list. It sets the selection by setting the File Selector's *GFSI_selection* field to the passed string.

If the File Selector is not on the screen or is suspended when it receives this message, it will set the selection and hold it until it scans the directory. The validity of the selection is not determined until the directory is again scanned. If the directory is not scanned immediately, the message's return value will be *true*. If the File Selector is on the screen and is not suspended, it will determine the validity of the selection immediately and return a value appropriate to the file's validity.

If the selection is valid, the user will see it highlighted in the File Selector's display. The selection will not be opened by this message; it is only selected.

Source: Unrestricted.

Destination: Any GenFileSelector object.

Parameters: *selection* A pointer to a character string giving the name of the file to be selected. The selection string is case-sensitive; if selecting a DOS file, it must be all upper case.

Return: An error flag: *false* if the selection is successfully made. The error condition can occur if the file is not found or if the File Selector can not currently be updated (it is not visible or is suspended).

Interception: Generally not intercepted.

■ MSG_GEN_FILE_SELECTOR_GET_SELECTION

```
dword   MSG_GEN_FILE_SELECTOR_GET_SELECTION(
        char    *selection);
```

This message returns the name, the entry number, and the entry flags of the file or directory currently selected.

Source: Unrestricted.

Destination: Any GenFileSelector object.

Parameters: *selection* A pointer to an empty character string at least **sizeof(FileLongName)** characters long. If just the entry number and entry flags are desired, pass a null pointer.

Return: The high word of the dword return value is the number of the selection in the File Selector's current file list. The low word is a record of type **GenFileSelectorEntryFlags**. Use GFS_GET_ENTRY_NUMBER to extract the entry number; use GFS_GET_ENTRY_FLAGS to extract the flags record.

selection The character string pointed to will be the null-terminated name of the current selection.

Interception: Generally not intercepted.

■ MSG_GEN_FILE_SELECTOR_SET_FULL_SELECTION_PATH

```
Boolean  MSG_GEN_FILE_SELECTOR_SET_FULL_SELECTION_PATH(
         char                *selection,
         DiskHandle          diskHandle);
```

This message causes the File Selector to set its path data and its *GFSI_selection* field according to the string passed in *selection*. The string's format is described below. The entry may be in any directory on any volume; the volume is specified by the passed disk handle.

If the File Selector is suspended or not currently on the screen when it receives this message, it will set the path and selection but will do nothing else until it becomes unsuspended or visible. Otherwise, the message will be handled immediately, and the File Selector will navigate to and scan the proper directory, setting the selection if possible.

If the path is valid but the selection is not, the File Selector will display the proper file list and set the selection to the first entry in the list. If any part of the path other than the selection is invalid, the File Selector will show the volume list.

Source: Unrestricted.

Destination: Any GenFileSelector object that is on the screen.

Parameters: *selection* A pointer to a character string containing the full path to be set as the File Selector's current path and selection. The path may be absolute or relative—if relative, it is considered relative to the File Selector's current directory and *diskHandle* will be ignored.

diskHandle The disk handle specifying the volume on which the new selection resides. If zero, the File Selector's currently-displayed disk will be used. If a relative path is specified in *selection*, this parameter will be ignored.

Return: This message returns an error flag: If the selection is successfully made, the flag will be *false*. If an error occurs, the flag will be *true*.

Interception: Generally not intercepted.

■ MSG_GEN_FILE_SELECTOR_GET_FULL_SELECTION_PATH

```
dword    MSG_GEN_FILE_SELECTOR_GET_FULL_SELECTION_PATH(
         char   *selection);
```

This message returns the disk handle and flags as well as the full pathname of the current selection.

Source: Unrestricted.

Destination: Any GenFileSelector object.

Parameters: *selection* A pointer to a character string at least PATH_BUFFER_SIZE characters long. Upon return, this string will contain the full selection path.

Return: The returned dword contains two word-sized fields: The high word represents the File Selector's current disk handle, and the low word is a **GenFileSelectorEntryFlags** record with the selection's flags. Use the macro GFS_GET_FULL_SELECTION_PATH_DISK_HANDLE to extract the disk handle, and use GFS_GET_ENTRY_FLAGS to extract the flags record. If sent when the File Selector is not visible on the screen, the message will return the disk handle and path of the last valid selection.

selection The character string pointed to will contain the full path and name of the current selection.

Interception: Generally not intercepted.

■ GFS_GET_FULL_SELECTION_PATH_DISK_HANDLE

```
DiskHandle GFS_GET_FULL_SELECTION_PATH_DISK_HANDLE(arg);
         dword  arg;
```

This macro extracts the disk handle (the high word) from a dword argument returned by MSG_GEN_FILE_SELECTOR_GET_FULL_SELECTION_PATH.

14.4.3 Rescanning Directories

MSG_GEN_FILE_SELECTOR_RESCAN,
MSG_GEN_FILE_SELECTOR_SUSPEND,
MSG_GEN_FILE_SELECTOR_END_SUSPEND

Each time the File Selector changes directories, it scans that directory and displays only the files it is supposed to. It looks into each file briefly to check the file against each criterion set in the File Selector. You can force the GenFileSelector to rescan the current directory with the message MSG_GEN_FILE_SELECTOR_RESCAN.

Occasionally you will want to scan the current directory only once after changing many parameters instead of rescanning after each change (search limitations, current volume, current selection, etc.). To optimize redrawing and rescanning, you can temporarily keep the GenFileSelector from scanning each time a MSG_GEN_FILE_SELECTOR_SET_... message is sent. To do this, send the following messages (both must be used, though not at the same time):

MSG_GEN_FILE_SELECTOR_SUSPEND
Notifies the File Selector that several operations in a row are coming and that it should suspend scanning the disk until the counterpart to this message (below) is received.

MSG_GEN_FILE_SELECTOR_END_SUSPEND
Indicates to the File Selector that directories may now be rescanned and displayed.

■ MSG_GEN_FILE_SELECTOR_RESCAN

```
void        MSG_GEN_FILE_SELECTOR_RESCAN();
```

This message causes the File Selector to rescan its current directory. The File Selector will re-build and re-display its entire file list. This message will only work when the File Selector is visible on the screen.

Source: Unrestricted.

Destination: Any GenFileSelector object that is on the screen.

Interception: Generally not intercepted.

■ MSG_GEN_FILE_SELECTOR_SUSPEND

```
Boolean  MSG_GEN_FILE_SELECTOR_SUSPEND();
```

This message causes the File Selector to suspend directory scans until a MSG_GEN_FILE_SELECTOR_END_SUSPEND is received. This provides the ability to avoid unnecessary directory rescans when several different attributes are being changed in succession (e.g. volume, mask, and file types).

No user action is affected by this message; only the following attribute-setting messages will be affected (their normal automatic rescan will not happen). Note that all messages have the prefix MSG_GEN_FILE_SELECTOR_:

SET_SELECTION
SET_MASK
SET_TOKEN
SET_GEODE_ATTRS
SET_FILE_CRITERIA
SET_FULL_SELECTION_PATH
SET_FILE_ATTRS
SET_CREATOR
SET_ATTRS

In addition to the above messages, the **GenClass** message MSG_GEN_PATH_SET will also be suspended.

Source: Unrestricted.

Destination: Any non-suspended GenFileSelector object.

Parameters: None.

Return: An error flag: *true* if the File Selector is already suspended, *false* if the suspension is successful.

Interception: Generally not intercepted.

Warnings: After suspending a File Selector with this message, be sure to unsuspend it with MSG_GEN_FILE_SELECTOR_END_SUSPEND.

■ MSG_GEN_FILE_SELECTOR_END_SUSPEND

```
Boolean  MSG_GEN_FILE_SELECTOR_END_SUSPEND();
```

This message unsuspends a File Selector that had previously been suspended with MSG_GEN_FILE_SELECTOR_SUSPEND. This message automatically causes a directory rescan. Because multiple suspension attempts are disallowed, this message only needs to be sent once to unsuspend the File Selector.

Source: Unrestricted.

Destination: Any suspended GenFileSelector object.

Parameters: None.

Return: An error flag: *true* if the File Selector was not previously suspended, *false* if the File Selector is successfully unsuspended.

Interception: Generally not intercepted.

14.4.4 Setting Scalable UI Data

```
HINT_FILE_SELECTOR_SCALABLE_UI_DATA
```

The File Selector lets you set which features you want based on the current application features, using HINT_FILE_SELECTOR_SCALABLE_UI_DATA. This hint will query the application object to get the current application features set, and it will set the File Selector's feature set (*GFSI_attrs*) accordingly.

The hint takes an array of **GenFileSelectorScalableUIEntry** structures, each of which has three fields:

GFSSUIE_command
: This field is a **GenFileSelectorScalableUICommand**, which is an enumerated type describing the effect of this particular entry. The five commands are detailed below.

GFSSUIE_appFeature
: This word represents the application's feature set or user level. The hint, when processed, will query the GenApplication object (the *GAI_appFeatures* value) and check those features against the features in this field. If they match, the hint will cause the features in *GFSSUIE_fsFeatures* to be set or cleared as appropriate.

GFSSUIE_fsFeatures
: This field represents the File Selector attributes that should be turned on or off depending on the command and the application features (see both fields above). The commands are defined below.

Following are the **GenFileSelectorScalableUICommand**s. Each queries the *GFSSUIE_appFeature* field and then acts appropriately.

GFSSUIC_SET_FEATURES_IF_APP_FEATURE_ON
This command will set the features in *GFSSUIE_fsFeatures* if the feature(s) in *GFSSUIE_appFeature* are turned on.

GFSSUIC_SET_FEATURES_IF_APP_FEATURE_OFF
This command will set the features in *GFSSUIE_fsFeatures* if the feature(s) in *GFSSUIE_appFeature* are off.

GFSSUIC_ADD_FEATURES_IF_APP_FEATURE_ON
This command will turn on the additional features in *GFSSUIE_fsFeatures* if the features in *GFSSUIE_appFeature* are turned on.

GFSSUIC_SET_FEATURES_IF_APP_LEVEL
This command will set the features in *GFSSUIE_fsFeatures* if the application is at the user level specified in *GFSSUIE_appFeature*.

GFSSUIC_ADD_FEATURES_IF_APP_LEVEL
This command will add the additional features in *GFSSUIE_fsFeatures* if the application is at the user level specified in *GFSSUIE_appFeature*.

Help Object Library

15

On-line help is important for many applications, especially in the world of graphical interfaces and consumer devices. Many users will opt for exploration of an application rather than reading through the manual, and on-line help can assist them.

The GEOS on-line help system is an information viewing tool to be used by most if not all GEOS applications. It is automated, and adding it to an application is simple. All you need to do is create your help documents with GeoWrite and then add a few help attributes to your application's generic UI objects.

15.1

Before using this chapter, you should know how to set up generic object resources (see "GEOS Programming," Chapter 5 of the Concepts Book), and be familiar with GeoWrite and its features.

15.1 What Is Help?

GEOS provides four basic flavors of help ranging from simple descriptions to a full-featured information viewer complete with linked text and embedded graphics. GEOS help is both context-sensitive and random-access; a user may get help on the current dialog box, for example, or may invoke your help system's table of contents directly.

How a user gets help depends on the system in use: A standard desktop PC will allow keyboard access with the F1 key as well as help triggers throughout the application's UI. A small-screen device may hide the help triggers and allow only a special "hard icon," special pen gesture, or other means of activation to save screen space. (Such decisions are up to the specific UI.)

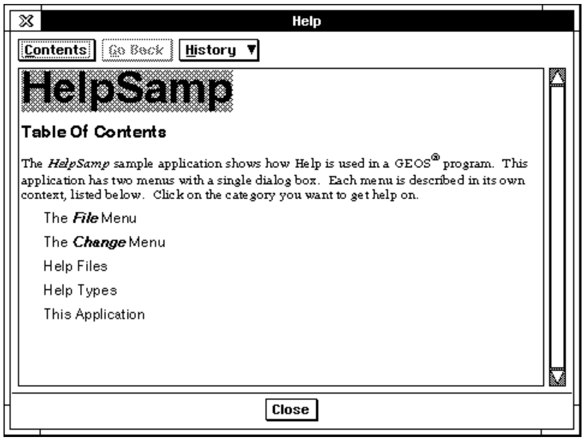
Nearly all applications will use "Normal Help," which provides a full-featured help viewer. The four types available are described below.

15.1

15.1.1 Normal Help

Most applications will include *Normal Help*, which provides a full-featured help viewer complete with embedded graphics and hyperlinked text. Normal help allows you to organize your help data in any manner; it has a single Table of Contents (TOC) page, which the user can access at any point in viewing help. Your TOC page should hyperlink to any other pages associated or appropriate (it depends on the information you're presenting). See Figure 15-1 for a sample of Normal Help.

Normal Help, like the other help types (see below), is also context-sensitive. You must define the contexts, but the help system will bring up the proper help page for the context. If the user clicks the help trigger on the primary window, for example, the TOC page is presented (by default). If the user clicks the trigger in a dialog box, however, the help page for that dialog box is presented.



HelpSamp

Table Of Contents

The *HelpSamp* sample application shows how Help is used in a GEOS® program. This application has two menus with a single dialog box. Each menu is described in its own context, listed below. Click on the category you want to get help on.

The ***File*** Menu

The ***Change*** Menu

Help Files

Help Types

This Application

Figure 15-1 *Normal Help*

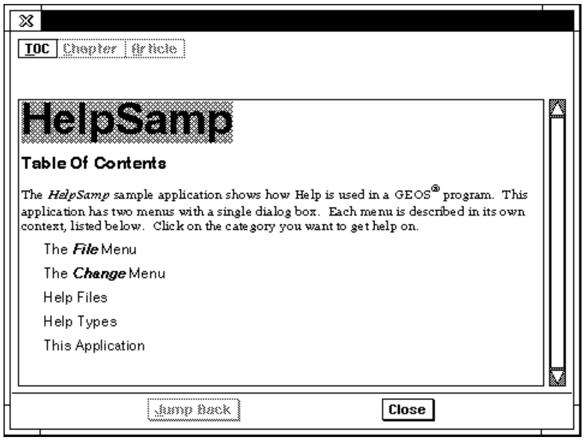
A sample screen of Normal Help. This is a TOC page from a help file defined for the HelpSamp application.

15.1.2 First Aid

First Aid help, like Normal Help, provides a full-featured help viewer. First Aid files are also organized by context, but the information is presented in a different format. Whereas Normal Help is organized around the structure of the application, First Aid is presented in a book format.

When a user clicks the First Aid trigger, the help page for the proper context is displayed; this will typically be taken from the current Focus object (as in Standard Help). From there, the user may navigate throughout the help "book" via both hyperlinks in the text and three buttons at the top of the help window: Contents, Chapter, and Article. An illustration of the First Aid help box is shown in Figure 15-2.

To use First Aid, an application needs its own HelpControl object. Simply add one as a child of your GenApplication and set its help type to HT_FIRST_AID. The help controller must also be put on the application's active list.



HelpSamp

Table Of Contents

The *HelpSamp* sample application shows how Help is used in a GEOS® program. This application has two menus with a single dialog box. Each menu is described in its own context, listed below. Click on the category you want to get help on.

The ***File*** Menu

The ***Change*** Menu

Help Files

Help Types

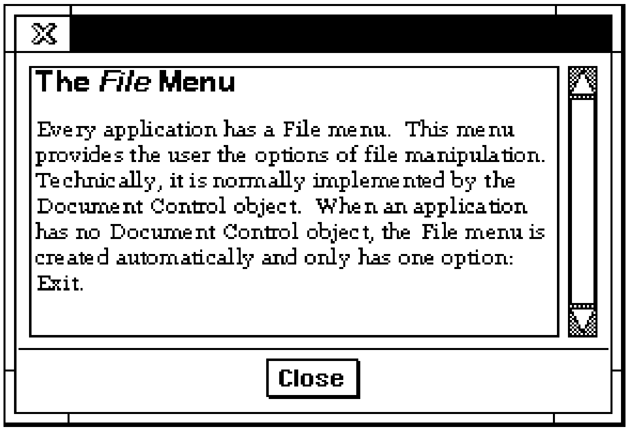
This Application

Figure 15-2 *First Aid Help*
A sample screen of First Aid Help.

15.1.3 Simple Help

Simple Help provides a small scrolling dialog box with help text. This is useful for places where very simple help is required, perhaps only a few lines to illustrate a simple operation. Simple Help may be most useful for appliance-type applications; most applications will use Standard Help or First Aid help. A sample screen of Simple help is shown in Figure 15-3.

15.2 Simple help, like First Aid, requires that the application have its own HelpControl object and that the help controller be on the active list. Text in simple help does not link, so all help text pertaining to the item must appear on a single page.



The *File* Menu

Every application has a File menu. This menu provides the user the options of file manipulation. Technically, it is normally implemented by the Document Control object. When an application has no Document Control object, the File menu is created automatically and only has one option: Exit.

Figure 15-3 *Simple Help*
A sample screen of Simple Help.

15.2 Adding Help to Your Application

For most applications, simply creating help files and adding ATTR_GEN_HELP_CONTEXT to certain generic objects will be sufficient. Other applications, though, such as those that use First Aid help, will require the addition of a HelpControl object (Normal help uses a system-provided and UI-run help controller).

The HelpControl object interacts with the **GenClass** and **MetaClass** aspects of generic objects to create help triggers, put up help screens, and determine which help file is used and what help context is displayed. For normal help, you don't need a HelpControl object; the system provides one. You do, however, have to know how to declare help contexts and work with help files.

15.2.1 Help Contexts and Help Triggers

ATTR_GEN_HELP_CONTEXT, HINT_PRIMARY_NO_HELP_BUTTON

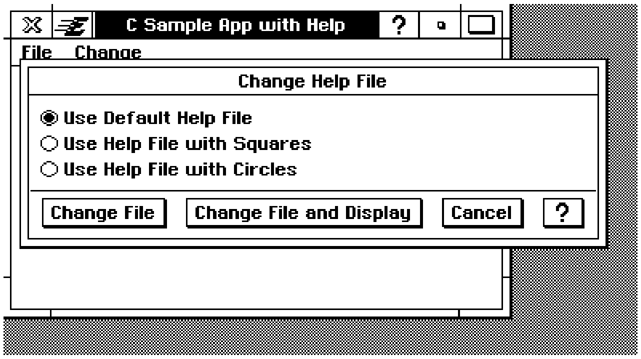
The help controller displays help pages based on an object's context. An object sets its context with ATTR_GEN_HELP_CONTEXT, which holds the name of the context. If an object does not have an assigned context, it will query up the tree until one of its ancestors assigns a context. The help control will display the help page with the same name as the assigned context.

This attribute has some side effects. If placed on a GenPrimary object, it will create the help trigger that appears in the application's title bar (unless other restrictions are applied—see below). If placed on a dialog box GenInteraction, it will create a help trigger in the dialog box (placement of the trigger is subject to the specific UI, but it will normally be on the far right of the reply bar). Both of these triggers invoke help with the context of their objects; both are illustrated in Figure 15-4.

Almost always, the GenPrimary's help context should be "TOC" to bring up the TOC help page. This is the default context, so if you don't specify a context for any object, help will always try to bring up the TOC page.

As stated above, ATTR_GEN_HELP_CONTEXT will by default create help triggers in certain situations. There are two times when creation of these help triggers will be suppressed: First, on small-screen devices or other situations wherein help triggers are not desired. In this situation, the .INI file will have a certain help option turned off. The simplest way to do this is to set

```
[uiFeatures]
helpOptions = 1
```



Change Help File

● Use Default Help File
○ Use Help File with Squares
○ Use Help File with Circles

Figure 15-4 *Help Triggers*

The Primary's help trigger and that in the dialog box are both created automatically by the HelpControl object.

in the .INI file. This corresponds to the single help option UIHO_HIDE_HELP_BUTTONS and will suppress the help triggers. Both the F1 key (or equivalent) and MSG_META_BRING_UP_HELP will still invoke help properly.

Second, you can suppress only the GenPrimary's help trigger by adding HINT_PRIMARY_NO_HELP_BUTTON to the GenPrimary. Other help triggers will appear as dictated by the other requirements.

You can create your own help triggers if you like, though there is rarely a need to. Help triggers can be normal GenTrigger objects, with special settings. If the trigger is in a dialog box, it should have IC_HELP as its interaction type (ATTR_GEN_TRIGGER_INTERACTION_COMMAND). Otherwise, it should send MSG_META_BRING_UP_HELP to the object that should provide the context for the help. Also, the trigger should have its moniker specified with ATTR_GEN_DEFAULT_MONIKER, with the value GDMT_HELP. This will ensure your help trigger looks like other help triggers.

Help triggers you create, however, will not be suppressed by the system. Thus, if you want your application to be adaptable to small-screen UIs, you should only use system-provided help triggers.

◆Objects

15.2.2 Adding Default Normal Help

To add normal help to your application, you only need to add ATTR_GEN_HELP_CONTEXT to the objects that should have help contexts. You can look at the HelpSamp sample application, though that does much more than you need. The code shown in Code Display 15-1 is taken from HelpSamp and shows a dialog box with a list, in which each entry has its own context.

You will want to put ATTR_GEN_HELP_CONTEXT on any generic object that could conceivably need help text. Nearly all applications will put help only on GenInteractions and GenPrimaries, but it is available for all generic objects. For example, items in a GenItemGroup may have help contexts (as shown in HelpSamp) so that when the user hits the F1 key, help will be invoked for that particular item. Likewise, menus and menu items may have their own help contexts.

Nearly all GenPrimary objects should have the context “TOC” set for them.

Code Display 15-1 Objects with Help Contexts

```
/*     The TypeDialog box has its own help context: Help Types. This will cause
 * the UI to put a help trigger in the dialog’s reply bar and, when that trigger
 * is clicked, will cause the system’s help object to bring up the normal help
 * viewer on the “Help Types” page in the current help file.
 *     Each item in the GenItemGroup has its own help context. The list itself
 * does not have a context because if the user presses the F1 key, the help system
 * will bring up the selected item’s help. If it were a list in which there could
 * be zero items selected, then the list might have its own context as well. */

@object GenInteractionClass TypeDialog = {
    GI_visMoniker = ’T’, “Change Help Type”;
    GI_comp = @TypeList;
    GII_visibility = GIV_DIALOG;
    GII_type = GIT_PROPERTIES;
    ATTR_GEN_HELP_CONTEXT = “Help Types”;
}

@object GenItemGroupClass TypeList = {
    GI_comp = @NormalItem, @FirstAidItem, @SimpleItem;
    GIGI_numSelections = 1;
    GIGI_selection = HT_NORMAL_HELP;
    GIGI_applyMsg = MSG_HELPSAMP_SET_TYPE;
    GIGI_destination = process;
    HINT_ORIENT_CHILDREN_VERTICALLY;
}

@object GenItemClass NormalItem = {
    GI_visMoniker = “Normal Help”;
    GII_identifier = HT_NORMAL_HELP;
    ATTR_GEN_HELP_CONTEXT = “Normal Help”;
}

@object GenItemClass FirstAidItem = {
    GI_visMoniker = “First Aid”;
    GII_identifier = HT_FIRST_AID;
    ATTR_GEN_HELP_CONTEXT = “First Aid”;
}

@object GenItemClass SimpleItem = {
    GI_visMoniker = “Simple Help”;
    GII_identifier = HT_SIMPLE_HELP;
    ATTR_GEN_HELP_CONTEXT = “Simple Help”;
}
```

15.2.3 Bringing Up Help on the Fly

`HelpSendHelpNotification()`

The Help Controller library provides a routine to bring up a help context, or to switch to a help context, at any time. This routine, **HelpSendHelpNotification()**, generates the proper notification event using the values you pass, then sends that notification to the help control object you’ve selected. This routine is detailed in the routines reference manual.

This routine becomes most useful when, for example, you want help on visible objects in a GenView or in a document. To call up a help viewer with a specific help file and context, you would have the visible object intercept both your special “help key” (e.g. the user clicks the middle mouse button) and MSG_META_BRING_UP_HELP, then call **HelpSendHelpNotification()**. In the handler for the input event, the object should *not* call the superclass. One example of this could be a special diagram of a floorplan, with each room a visible object; each “room” would bring up a different help context describing the characteristics of the room.

15.3 Customizing Help

If you want to use one of the other help types, or if you want to use multiple help types in a single application, you must do more than just apply help contexts to objects. You can customize several things about the help system:

- Help Type
  You can set or change the type of help offered by your application. You must have one HelpControl object for each type of help offered. You can set different objects to display or use different help types.
- Help File
  Any object can specify the name of the file from which it draws its help. You can set the help file statically, change it dynamically, or have it defined by the GEOS.INI file.
- Pointer Image
  The help system lets you define a custom pointer image to be used when the mouse pointer is over a hyperlink. You define the image, and the help controller uses that instead of the standard link pointer.
- Help Features
  Because the HelpControl object is a controller, it has a certain feature set that may be turned on and off with ATTR_GEN_CONTROL_REQUIRE_UI and ATTR_GEN_CONTROL_PROHIBIT_UI. The features include the buttons presented to the user for navigating the help file and for closing the dialog box.

You can even create special “viewer” applications based on the help object. To make any of the above customizations, however, you must add one or more HelpControl objects to your application.

15.3.1 Bringing Up Initial Help

`ATTR_HELP_INITIAL_HELP_FILE, ATTR_HELP_INITIAL_HELP`

An object may use ATTR_HELP_INITIAL_HELP_FILE and ATTR_HELP_INITIAL_HELP to bring up a help context when the object becomes usable. ATTR_HELP_INITIAL_HELP_FILE defines the help file from which the help is to be taken; it is only used if ATTR_HELP_INITIAL_HELP is also present. This one defines the help context to be brought up on startup.

These attributes are used normally only by help-viewer applications, as it is unlikely help will be required on startup. Normally, help is brought up by the user pressing a help trigger.

15.3.2 Adding the HelpControl

To add a help controller to your application, simply declare an instance of **HelpControlClass** and add it as a child of your GenApplication, adding it to the active list. The code in Code Display 15-2 is taken from the HelpSamp sample application and illustrates what you have to do to support custom help. The steps are

◆ Declare the HelpControl
Declare an object of **HelpControlClass**. You must set its *HCI_helpType* field to the **HelpType** value it will manage. Because the standard behavior is to have the help window always appear on top of other dialog boxes, you should set the window priority accordingly (as shown). Also, because most controllers will come up disabled, set the GS_ENABLED flag to have the help buttons enabled when help is first invoked.

◆ Add the HelpControl to the active list
The HelpControl, as all controllers, must be put on the active list, the MGCNLT_ACTIVE_LIST GCN list. All of your help controllers must be on this list.

◆ Set the help type in the GenApplication
If you're simply using a different help type, set it in the GenApplication object with ATTR_GEN_HELP_TYPE. The sample code shown does not have this attribute because the application uses multiple types but defaults to Normal Help. The user has the option of changing the help

type; when the type is changed, ATTR_GEN_HELP_TYPE is added to the GenApplication dynamically.

Other than the above customizations, using another help type is as simple as using Normal Help. Other considerations must be observed when creating the help files, but the code difference is simple.

Code Display 15-2 Adding Help Controllers

```
/*    The GenApplication has two help controllers as its children.
 * FirstAidHelpControl manages First Aid help, and SimpleHelpControl manages
 * Simple Help. In addition, the Normal Help controller—provided by the
 * system—manages Normal Help. Both custom help controllers must be put both on
 * the active list and on the GAGCNLT_NOTIFY_HELP_CONTEXT_CHANGE list.
 *    Normally, the GenApplication would have ATTR_GEN_HELP_TYPE declaring the
 * help type used. This application, however, defaults to HT_SYSTEM_HELP and
 * therefore does not need the attribute. */

@object GenApplicationClass HelpSampApp = {
    GI_visMoniker = list { @HelpSampTextMoniker };
    GI_comp = @HelpSampPrimary, @FirstAidHelpControl, @SimpleHelpControl;
    gcnList(MANUFACTURER_ID_GEOWORKS,GAGCNLT_WINDOWS) = @HelpSampPrimary;
    gcnList(MANUFACTURER_ID_GEOWORKS, MGCNLT_ACTIVE_LIST) =
                                @FirstAidHelpControl, @SimpleHelpControl;
}

@visMoniker HelpSampTextMoniker = “C Sample App with Help”;

/*    The help controllers may manage at most one HelpType each. Thus, you must
 * set the HCI_helpType field so the controller knows what type of help it
 * manages. */

@object HelpControlClass FirstAidHelpControl = {
    GI_states = @default | GS_ENABLED;
    HCI_helpType = HT_FIRST_AID;
}

@object HelpControlClass SimpleHelpControl = {
    GI_states = @default | GS_ENABLED;
    HCI_helpType = HT_SIMPLE_HELP;
}
```

15.3.3 Sizing the Help Dialog Box

`HINT_HELP_TEXT_FIXED_SIZE, HINT_HELP_NOT_RESIZABLE`

Depending on what you're using help for, you may want to keep the help dialog box a fixed size, or you may wish to make it not resizable. For both cases, you must use a custom help controller object and set one of the following hints on the controller:

HINT_HELP_TEXT_FIXED_SIZE
This hint is exactly the same as **GenClass**' HINT_FIXED_SIZE; it takes a **CompSizeHintArgs** structure as its extra data and sets the help controller's dialog box to the fixed size.

HINT_HELP_NOT_RESIZABLE
Normally, help controllers set up as GIV_DIALOG are resizable. This hint will counteract that behavior and make the window not resizable.

15.3.4 Managing Help Types

`HelpType, ATTR_GEN_HELP_TYPE, MSG_META_GET_HELP_TYPE`

The brand of help your application provides is defined by its **HelpType**. This value is set in the HelpControl object's *HCI_helpType* instance field. For normal help, you don't need to set anything; the type will automatically be HT_SYSTEM_HELP for the system help object (you should not set this type for your own help control objects).

There are five other help types you can use: HT_NORMAL_HELP, HT_FIRST_AID, HT_STATUS_HELP, HT_SIMPLE_HELP, and HT_SYSTEM_MODAL_HELP. You should only explicitly use HT_NORMAL_HELP if you want the normal help setup with some customizations or you are creating a viewer application. The others you must use explicitly if you want to provide those types of help.

(Note: HT_SYSTEM_MODAL_HELP is used for providing help within system-modal dialog boxes only. It is rare that an application will ever use system-modal dialog boxes, but if yours does and you need to provide help within it, use this special type. Such dialog boxes should be given the attribute ATTR_GEN_HELP_TYPE with this type to distinguish them from other dialogs.)

To set an application's help type, you have to first supply a HelpControl object as a child of the GenApplication and set its *HCI_helpType* field to the proper help type. You also have to put your HelpControl object on the GenApplication's GAGCNLT_NOTIFY_HELP_CONTEXT_CHANGE GCN list. You may have one HelpControl object for each help type supported by your application.

Examples of how to set up the proper objects are shown above in Code Display 15-2 on page ◆ 1003.

Any object may have any help type associated with it. You could have, for example, one dialog box set to Normal Help and another set for Simple Help. This is unusual, and you will usually want to stick to a single help type within one application. To set an object's help type, use ATTR_GEN_HELP_TYPE. You can retrieve the current help type of an object by sending it MSG_META_GET_HELP_TYPE.

15.3.5 Managing Help Files

ATTR_GEN_HELP_FILE, ATTR_GEN_HELP_FILE_FROM_INIT_FILE, MSG_META_GET_HELP_FILE, MSG_META_SET_HELP_FILE, ATTR_GEN_CONTROL_DO_NOT_USE_LIBRARY_NAME_FOR_HELP

The help controller gets the text it displays from help files. Help files are generated by the help editor (a modified GeoWrite) from GeoWrite documents and are located in USERDATA\HELP. An application may use a single help file or multiple help files, and hyperlinks may extend across files.

The help controller figures out the help file to use by looking for ATTR_GEN_HELP_FILE on the generic object that brings up the help. For example, if the user clicks a help trigger in a dialog box, the help controller first gets the context of the dialog box (contexts are described in "Help Contexts and Help Triggers" on page 997) and will then get its help file. If the dialog box has an ATTR_GEN_HELP_FILE specifying a particular help file, the help controller will try to use that file.

If the dialog box does not have that attribute, the help controller queries up the generic tree until it either finds an ancestor with the attribute or gets to the GenApplication object. If no help file is specified, the help controller will by default use a file with the same name as the geode's permanent name with the extender characters removed. For example, in the sample application HelpSamp, which has the permanent name *helpsamp.app* (which is defined in the **helpsamp.gp** file), the default file name would be **helpsamp**.

15.3

An object can use ATTR_GEN_HELP_FILE_FROM_INIT_FILE to specify that its help file should be garnered from the GEOS.INI file. If this attribute is used, the help controller will look in the GEOS.INI file for a category of the same name as the application and a key named "helpfile." Thus, to set the help file to "My Own Help File" for the HelpSamp application, you could add the following to your GEOS.INI file:

```
[HelpSamp]
helpfile = My Own Help File
```

If, however, no object has ATTR_GEN_HELP_FILE_FROM_INIT_FILE, this entry will not be noticed by the help controller.

Controller objects, which typically exist within their own libraries, may use the help files provided by the application that includes them rather than the help file normally used by the library. To have a controller do this, set ATTR_GEN_CONTROL_DO_NOT_USE_LIBRARY_NAME_FOR_HELP for the controller in your application. This will have the controller continue querying up the generic tree when searching for the proper help file to use.

The help file used by a given object may also be changed dynamically with MSG_META_SET_HELP_FILE. This message takes a pointer to the new file's name and sets the object's ATTR_GEN_HELP_FILE attribute to the new file. This message is used by the HelpSamp sample application.

The help file can be retrieved with MSG_META_GET_HELP_FILE. This message returns the name of the file being used by the recipient generic object.

15.3.6 Customizing the Pointer Image

ATTR_HELP_CUSTOM_POINTER_IMAGE,
MSG_HELP_CONTROL_GET_POINTER_IMAGE

The help controller changes the pointer image when the pointer is over a link in the help text. This is so the user knows where to click and expects a link to be followed. Applications that want custom pointers can set the "link" pointer image with ATTR_HELP_CUSTOM_POINTER_IMAGE. This attribute takes an optr to a custom pointer image definition.

The pointer image is a **PointerDef** structure and must be stored in a chunk in a sharable, non-duplicated data block. Note also that to use a custom pointer image, you must explicitly declare a HelpControl object, even for Normal Help.

You can retrieve the link pointer image by sending MSG_HELP_CONTROL_GET_POINTER_IMAGE to the HelpControl object. This will return the current image in use, either a custom image or the default. This message is detailed on page 1017.

15.3.7 Changing the Help Features

Because the HelpControl is a subclass of **GenControlClass**, you can change the features it offers both in your **.goc** file and dynamically. For complete information, see section 12.1.1 of chapter 12. The features of the help controller are detailed in "HelpControlClass Reference" on page 1014.

15.4 Creating Help Files

Perhaps the most involved step of adding help to your application is actually creating the help files. Help files are special, compressed versions of GeoWrite files that reside in the \USERDATA\HELP directory. There are several steps to creating help files:

1 Enable the Help Editor
The Help Editor is simply a special version of GeoWrite.

2 Organize and write the text
Depending on your application, your help text, and how you wish to present it, this can be simple or complex. You must keep several things in mind when organizing your help.

3 Define files and contexts
Each help file must have its contexts named, and each context must be defined. If you are using multiple help files, you must define the help contexts for each file in use, and you must "define" each file being used.

4 Set contexts
Each page of help must have its own context name.

5 Set hyperlinks
After all contexts are named and set, you may set hyperlinks to them.

6 Generate the help files
When the help text is ready, you must generate the help file(s). They will then be available for your application to open and view.

15.4.1 Enabling the Help Editor

The Help Editor is GeoWrite with the Help Editor option turned on. To turn on this option, manually edit your .INI file to add the category and key as follow:

```
[configure]
helpEditor = true
```

Without the above lines, you can't access the Help Editor. After adding them, start up GeoWrite and change the user level. Choose "Customize" and turn on the Help Editor option at the end of the list. When you apply the change, a new "Help Editor" menu should appear towards the right of the menu bar.

15.4.2 Organizing and Writing the Text

If your help text is simple and straightforward, you can probably dive right in and write it. If it is not, though, you should spend some time up front organizing it, naming the contexts and files, and figuring out where hyperlinks will be placed. Defining contexts and links as you're writing can quickly get confusing, even for experienced writers.

Most help text will be configured for Normal Help. This type of help typically has a table of contents page (TOC) and numerous other contexts, many of which are linked to each other. Normally, a help context (page) will be accessible through two means: First, the user could click a help trigger or hit F1 at the right time to bring up the proper context. In this case, the user can not "go back" to the previous context, although he can get to the TOC page by clicking the Contents button. Second, the user could navigate through the help system via hyperlinks and get to the context.

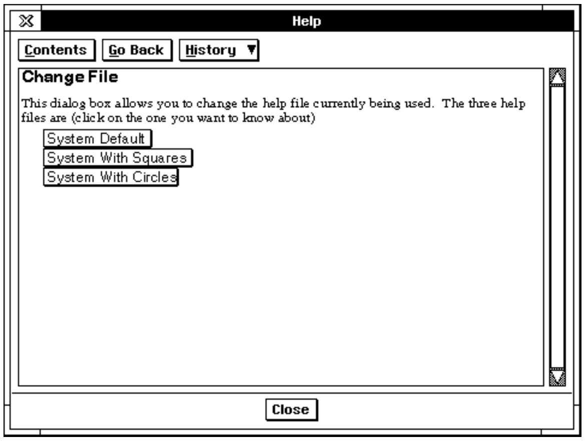
Not all contexts may be available through links, however; some may be accessible only through clicking a help trigger or hitting F1. Other contexts may be accessible only via navigation of hyperlinks in the help document. You should plan this out in advance so you can write effective text and set your links properly.

15.4.2.1 Organizing the Help

Perhaps the most straightforward help organization is a simple listing of topics, each of which is a hyperlink to another help screen. For example, the "Change File" help screen in the HelpSamp sample application's default file gives a brief description of the Change File dialog box and then three subtopics, each of which links to another context in the help file. (This screen is shown in Figure 15-5.)

A First Aid file, however, is designed differently. It has three "levels" of help: Contents, Chapter, and Article. The Contents page is a TOC page, just as in Normal Help. The TOC button allows the user quick access to that level at all times. When the user selects an entry on the TOC page, a Chapter-level context should be brought up. At this point, only the TOC button is enabled, and the Chapter button is enabled and selected.

The Chapter-level context should similarly list subtopics of interest. When the user selects one of these, an Article-level context will be displayed. At this point, both the Contents button and the Chapter button will be enabled, allowing the user to "back up" to one of those previous levels. Also at this point, the Article button will be enabled and selected. The Article-level context (currently viewed) may contain a list of several questions or subtopics. When the user follows one of these links, the Article button will be enabled but deselected, allowing the user to quickly return to any of the three levels traversed to this point.



Change File

This dialog box allows you to change the help file currently being used. The three help files are (click on the one you want to know about)

System Default
System With Squares
System With Circles

Figure 15-5 *A Sample Help Context*

This page has a brief description with a list of subtopics, each of which is a link to another help context.

First Aid, like Normal Help, may have contexts linked to any other contexts in the same or other files; they are not restricted to the TOC–Chapter–Article links mentioned above.

Simple Help normally will not have any links because it offers no way to go back or even to return to a TOC page. If your application offers Simple Help, you should keep it straightforward and not put in any links.

15.4.2.2 Using Visual Cues and Graphics

Although the help controller will change the pointer image when the pointer is over a hyperlink, you should use graphics or highlighted text whenever possible to call attention to the links. Because the Help Editor is extended GeoWrite, anything you can do to normal GeoWrite text you can put in your help files.

Long pages of text can be difficult to read unless you put in headers or other highlighting. Some suggested highlighting techniques are to use colored or gradient-filled text, use the boxed or button text styles, or underline the text.

Using different sizes and fonts is also effective. Keep in mind, however, that many users will have monochrome, CGA-style screens.

You can also embed graphics in the text. This does not include using the "graphic layer" of GeoWrite, however—since the Help Editor uses only the text layer, it will not use graphics pasted to the graphic layer. You must paste them directly into the text.

A useful tip to creating consistent help documents is to pre-define a number of style sheets in GeoWrite and use those styles for all your help. Any time a new style is required, create a new style sheet.

15.4.3 Defining Files and Contexts

If all your help resides within single files and links do not cross files (all links are internal to their own files), then defining contexts is simple. However, if links can cross files, you must do a little more work.

When you "define a context," you are notifying the Help Editor that a context with a given name and type exists in the specified file. When all links are internal to a single file, or when you have no links (as with Simple Help), you can simply define contexts for the "same file." However, if you have links crossing files, you must define both the file and context for each external link (similar in function to an "external" definition in your code).

To Define a File

Open the Define File dialog box via the Help Editor menu. Type the name of the file that will have links into it, and click "Add File." You can delete or rename previously defined files.

To Define a Context

Open the Define Context dialog box via the Help Editor menu. Use the upper-left popup list to select which file the new context will be defined for. Use the upper-right popup list to define the type of the file:

Text A normal text help page. Use this for all contexts in all help types except First Aid as noted below.

TOC A First Aid TOC page.

15.4

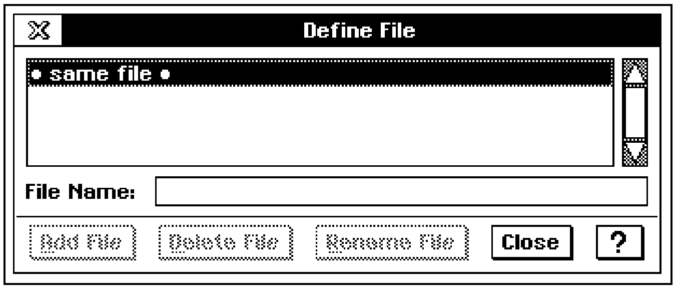


Figure 15-6 *The Define File Dialog*

Define each file linked to. You can't set a link to a context unless it has been defined for the file in which it resides.

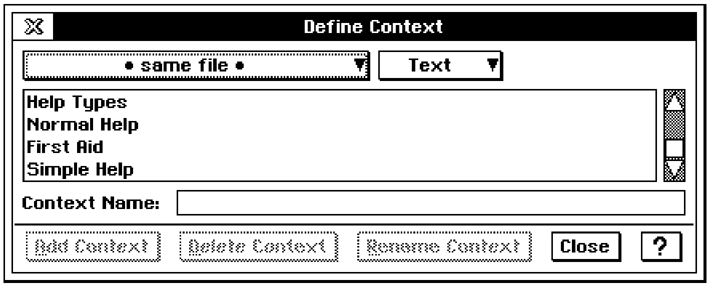


Figure 15-7 *The Define Context Dialog*

Define contexts for each file linked to. You can't set a link to a context unless it has been defined for the file in which it resides.

Chapter A First Aid Chapter-level page.

Article A First Aid Article-level page.

Once you've selected the proper type, name the context and click “Add Context.” You can also delete and rename previously-defined contexts.

To Set a Context

Each help page may be recognized as only one context. The context for the page must be set at the page's beginning. To set a context, first open the “Set Context” dialog box via the Help Editor menu. Then select some text at the beginning of the help page, select the context name from the dialog box, and click “Apply.” That page will be set to the selected context.

◆Objects

Pages in the GeoWrite file that do not have help contexts applied will not be generated into the help file. This fact is useful for recording information within the file about the file itself (release dates, draft dates, author name, revision notes, etc.).

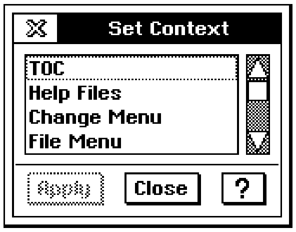


Figure 15-8 *The Set Context Dialog*

Set a page's context at the very beginning of the page. Only the context at the beginning is recognized.

15.4.4 Using Hyperlinks

You can set any selected text or graphic to be a hyperlink to another context in any help file. To set a hyperlink, open the "Set Hyperlink" dialog box via the Help Editor menu. Select the text that, when clicked on, should cause the new context page to appear. Then select, in the "Set Hyperlink" dialog, the appropriate file and context names. Click "Apply," and the selected text and/or graphic will act as a hyperlink to that file and context.

Once a hyperlink is created, you can follow it in the GeoWrite document by using the "Follow Hyperlink" entry in the Help Editor menu. This option will not be enabled unless the current cursor or selection is a hyperlink.

15.4.5 Generating the Help Files

Once the help file is created, all the contexts are defined and set, and all the hyperlinks (if any) are established, you can generate the help file. When the file is generated, it is automatically placed in USERDATA\HELP, and it will have the same name as the GeoWrite document being edited. Thus, if you are editing a file called "helpsamp," the help file would be USERDATA\HELP\helpsamp.

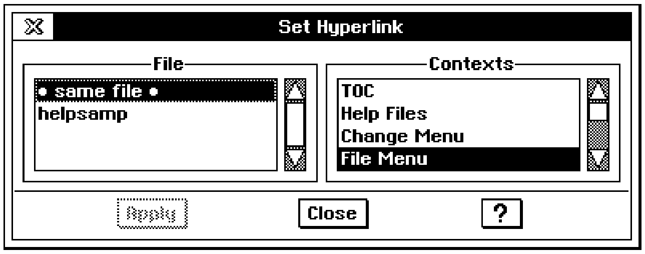


Figure 15-9 *The Set Hyperlink Dialog*

Set a hyperlink to the file and context selected.

The help file may be compressed or not. The default is to compress the help data to save disk space. Since help files are read-only, the help controller decompresses the data when loading it, then simply discards it when done. The “Compress Data in File” option in the Help Editor menu allows you to specify whether the help file should be compressed or not (just about the only time it should not be compressed is for Status Help).

To generate the help file, simply select “Generate Help File” in the Help Editor menu. The generation is one-way; you can *not* edit a help file. You should, therefore, keep your original GeoWrite documents containing the help text available in case they require editing or update.

15.5 HelpControlClass Reference

This section will only be useful if you plan on customizing help for your application. Nearly all of what you need for creating normal help files and adding help to your application is described in the previous sections. The following is detailed reference information for **HelpControlClass** (shown in Code Display 15-3).

Code Display 15-3 HelpControlClass

```
        @instance HelpType      HCI_helpType;            /* See HelpType, below */
        @instance MemHandle     HCI_curFile;             /* Internal */
        @instance MemHandle     HCI_historyBuf;          /* Internal */
        @instance word          HCI_nameArrayVM;         /* Internal */
        @instance GeodeHandle   HCI_compressLib;         /* Internal */
            @default GII_attrs = (@default | GIA_NOT_USER_INITIATABLE);
            @default GII_visibility = GIV_DIALOG;

        @vardata void                   ATTR_HELP_SUPPRESS_INITIATE;
        @vardata char[]                 ATTR_HELP_INITIAL_HELP;
        @vardata optr                   ATTR_HELP_CUSTOM_POINTER_IMAGE;
        @vardata CompSizeHintArgs        HINT_HELP_TEXT_FIXED_SIZE;
        @vardata void                   HINT_HELP_NOT_RESIZABLE;
        @vardata char[]                 ATTR_HELP_INITIAL_HELP_FILE;

typedef ByteEnum HelpType;
    #define HT_NORMAL_HELP      0
    #define HT_FIRST_AID        1
    #define HT_STATUS_HELP      2
    #define HT_SIMPLE_HELP      3
    #define HT_SYSTEM_HELP      4       /* Reserved for system use */

/* Internal Vardata fields—do not use these. */
        @vardata char[]         TEMP_HELP_ERROR_FILENAME;
        @vardata char[]         TEMP_HELP_TOC_FILENAME;
        @vardata void           TEMP_HELP_DETACH_RECEIVED;
```

HelpControlClass is a subclass of **GenControlClass** and, as such, inherits the feature management attributes and messages thereof. For complete information on feature and tool sets, see section 12.1 of chapter 12. The HelpControl object has features but no tools. The feature set of **HelpControlClass** is shown in Code Display 15-4.

Code Display 15-4 HelpControl Features

```
/* These features can be turned on or off with ATTR_GEN_CONTROL_REQUIRE_UI and
 * ATTR_GEN_CONTROL_PROHIBIT_UI. They can be turned on or off dynamically with
 * MSG_GEN_CONTROL_ADD_FEATURE and MSG_GEN_CONTROL_REMOVE_FEATURE. */
```

```
typedef WordFlags HPCFeatures;
    #define HPCF_HELP                   0x0100  /* has a “help on help” trigger */
    #define HPCF_TEXT                   0x0080  /* has text */
    #define HPCF_CONTENTS               0x0040  /* has “Contents” button */
    #define HPCF_HISTORY                0x0020  /* has “History” button */
    #define HPCF_GO_BACK                0x0010  /* has “Go Back” button */
    #define HPCF_CLOSE                  0x0008  /* has “Close” button */
    #define HPCF_INSTRUCTIONS           0x0004  /* has “Instructions” button */
    #define HPCF_FIRST_AID_GO_BACK      0x0002  /* has “Go Back” button for
                                                 * First Aid */
    #define HPCF_FIRST_AID              0x0001  /* has First Aid configuration */
```

HelpControlClass, as a GenControl subclass, receives notification when the controlled item changes. For example, a change in help context will cause the HelpControl object to display the new context page. The HelpControl object must be put on the GAGCNLT_NOTIFY_HELP_CONTEXT_CHANGE notification list. When a context change occurs, the help controller will receive MSG_META_NOTIFY_WITH_DATA_BLOCK, with the notification type GWNT_HELP_CONTEXT_CHANGE. This notification type passes a data block containing a **NotifyHelpContextChange** structure, which is shown below.

Code Display 15-5 Help Notification Structure

```
/* This data structure is passed in MSG_META_NOTIFY_WITH_DATA_BLOCK to the help
 * controller when a change in the help context occurs. This structure can be
 * generated by calling HelpSendHelpNotification(). This structure uses the
 * following constant and type. */

#define MAX_CONTEXT_NAME_SIZE    20

typedef char    ContextName[MAX_CONTEXT_NAME_SIZE];

typedef struct {
    HelpType            NHCC_type;              /* HelpType involved in change */
    ContextName         NHCC_context;           /* New context to link to */
    FileLongName        NHCC_filename;          /* New file name to link to */
    FileLongName        NHCC_filenameTOC;       /* File name to get TOC from */
} NotifyHelpContextChange;
```

Objects

HelpControlClass has two exported messages that subclasses may intercept. These are not often intercepted by subclasses. Their reference information is presented below.

■ MSG_HELP_CONTROL_FOLLOW_LINK

Cause the HelpControl object to follow the specified link in the help file.

Source: Unrestricted; typically internal to the controller.

Destination: The HelpControl object to follow the link.

Parameters:	*link name*	16-bit token of the link name.
	link file	16-bit token of the link file.

Return: Nothing.

Interception: Not generally intercepted.

■ MSG_HELP_CONTROL_GET_POINTER_IMAGE

Return the pointer image to be used when the pointer is over a link in the help text. It looks for ATTR_HELP_CUSTOM_POINTER_IMAGE and returns the pointer stored there; if no custom pointer exists, the default is returned.

Source: Unrestricted; typically internal to the controller.

Destination: The HelpControl object using the custom pointer.

Parameters:	*not over link*	Pass FALSE if the pointer is *not* over a link, TRUE if it is. (Must pass TRUE, not just non-zero).

Return: A word of **MouseReturnFlags**. If MRF_SET_POINTER_IMAGE is set in this record, the optr of the pointer image will also be returned. Otherwise, the optr will be NullOptr.

Interception: A “viewer” application may subclass this and return its own pointer images if they are dynamic. If static pointer images are used, the application should use ATTR_HELP_CUSTOM_POINTER_IMAGE instead. There is no need to call the superclass with this message.

15.5

Objects

Impex Library

16

Most applications which write data files should be compatible with other applications. For example, applications will want to be able to *import* data, i.e. open a file written by another application and translate its data into a format the application can use. However, writing code to open another application's file and translate it into your own application's format can be difficult. Any application that wanted to properly support file-importing would have to provide a great many of these utilities. Furthermore, there would be a lot of duplicated effort; one application's routine for importing a particular file format would be much like another's. Applications would have similar problems *exporting* data, i.e. writing their data in another application's format.

The Impex (Import/Export) Library, along with specific translation libraries, solves these problems. They automatically translate files from non-GEOS formats into the *Metafile* (i.e. Clipboard) format and back. Applications just need to include the Import and Export objects; once they do this, importing and exporting data is just like using the Clipboard. Furthermore, extending the Impex Library to handle new formats is easy. All you have to do is write a new translation library for that one format and put the library in the SP_IMPORT_EXPORT_DRIVERS standard path. All applications which use the Impex objects will then be able to use that new format. Developers can write their own translation libraries; however, few will need to do this, since GEOS provides libraries for the most popular formats.

The Impex library currently supports import and export of text and graphic files. Geoworks is planning on extending it to allow the import and export of spreadsheet and font files as well. When these capabilities are available, they will be just like the already-supported text and graphic import and export.

The Impex Library works very much like the Clipboard. For this reason, you should be familiar with the Clipboard (see "The Clipboard," Chapter 7 of the Concepts Book). You should also be familiar with VM chains (described in "Virtual Memory," Chapter 18 of the Concepts Book).

16.1 Impex Basics

The Impex library provides a uniform way of importing data from files written by non-GEOS applications, and of exporting data to files written in these formats. It also saves coding time. Every outside format needs a single translation library, which translates between the Metafile format and the format for that specific library. The Impex objects provide all the necessary user interface (though the libraries and applications can add more), and also do all necessary interaction with the translation libraries.

In order to import data, an application will need to have an ImportControl object. Similarly, in order to export data, an application will need an ExportControl. Most applications that produce data files will want to use both of these objects.

Some applications will create a new GEOS file for imported data; others will add it into the current document, as if they were pasting data from the clipboard. Similarly, some applications will export the entire current document; others will export only the current selection, if any. This is left entirely to the discretion of the application.

16.1.1 The Impex Objects

There are two Impex objects which applications need to know about: ImportControl and ExportControl. These objects are both subclassed off of **ImportExportClass**, which provides some functionality needed for both objects; however, all this functionality is internal, so you can ignore this class. (No one should ever create an ImportExport object, and it has no instance data which applications may set.) **ImportExportClass** is itself subclassed from **GenControlClass**; therefore, ImportControl and ExportControl objects inherit all the functionality of controllers.

Applications will generally have one ImportControl and one ExportControl object. Applications may put these in different places; for example, some will place both of these on the File menu, while others may put them in the "Open" or "Save As" dialog boxes. Applications should decide this based on how they think the user will be using the import and export capabilities.

The ImportControl and ExportControl objects communicate with the application by sending messages. The ImportControl object sends a message to its recipient near the end of the Import process, when it has translated a file into a VM chain for the application to copy. The ExportControl object sends a message near the beginning of the export process, instructing the application to create a VM chain for the application to export. In both cases the message sent and the recipient are set by the application. Most applications will have the messages sent to their process arguments; however, applications which use the object model of document control may want to have the messages sent to the target object.

16.1.2 How the Impex Objects Work

The Impex objects manage the user interface for importing and exporting data. They also open and create the appropriate files when necessary. If a user wants to import a file, the Import object lets the user select a source file and format. The import object will then call an appropriate library to translate the source file into a VM chain in the clipboard (i.e. MetaFile) format. At this point, it sends a message to the application, passing the handle of the VM chain. Similarly, when the user wants to export data, the Export object first lets the user select a name and format for the destination file. The Export object then sends a message to the application requesting a source VM chain in the Metafile format. The Export object then calls the appropriate translation library, which translates the MetaFile into a corresponding DOS file in the specified format.

It may help to give a blow-by-blow example. Let's suppose someone is using FooPaint, a graphics program for GEOS. FooPaint uses both the ImportControl and ExportControl objects to let the user import and export graphics. The user wants to import a graphic file written by BazDraw, a non-GEOS application. The user activates the "Import" trigger. The following things happen:

1 The ImportControl checks what translation libraries are available. This tells it what formats can be imported.

2 The ImportControl presents a dialog box to the user. This box contains a list of available formats. It also contains a file selector, which the user uses to select a file of the appropriate format. Assume the user chooses the BazDraw format, and the file FISH.BAZ.

3. The ImportControl opens FISH.BAZ and a temporary VM file. It then starts up the BazDraw-to-Metafile library. It then sends a message to this library, passing both files' handles with the message.
4. The library reads FISH.BAZ and writes an equivalent Metafile sequence in a VM chain in the temporary file. When it is done, it returns the VM chain handle to the ImportControl. 16.1
5. The ImportControl sends a message to its destination object (often the application's Process object). This is the application's first direct involvement in the import. The message includes such information as the handles of the VM chain, the type of data being imported (in this case, a graphics metafile), and the message to send on completion.
6. The application copies the information from the VM chain. This is almost identical to pasting data from the clipboard. FooPaint will probably respond by adding the graphic to the target document.
7. When the application is finished copying the data, it calls the **ImpexImportExportCompleted()** utility routine. This routine sends an appropriate acknowledgment message to the ImportControl object. The ImportControl destroys the temporary VM file.

Note that almost all of this takes place without any action by the application. The only thing the application has to do is copy data from the VM chain to its own file, translating from the Metafile format to its own format. This is exactly what the application does whenever it pastes data from the clipboard; thus, applications which use the clipboard already have almost everything they need to use the Impex objects.

Exporting data is almost the same as importing it. The application is sent a message at the beginning of the operation, asking the application to write a VM chain in the Metafile format. The ExportControl will already have created a temporary VM file for the application to use; the application just has to allocate a chain in it and fill it with the appropriate information. The ExportControl presents a dialog box to the user, letting the user select a name, format, and location for the exported file; it then starts up the appropriate translation library to write the file.

◆Objects

16.2 Using Impex

The Impex objects are easy to use. Essentially, an application has to declare two objects and handle a single message from each of them. Applications which can use the clipboard already have most of the code they will need to use the Impex objects.

16.2.1 Common Impex Concepts

There are certain concepts and structures which are used by both of the Impex objects. Applications which use the Impex objects will have to be familiar with them.

16.2.1.1 Metafile Formats

`ImpexDataClasses`

The Impex objects are designed to translate into a few specific formats. These formats are known collectively as the *Metafile* formats. These formats (except for the "font" format) are identical to the standard "Clipboard Item Formats" described in the Clipboard chapter (see section 7.2.4 of "The Clipboard," Chapter 7 of the Concepts Book). When an application uses the Impex objects, it has to specify what types of data it is prepared to import or export. It does this by setting an **ImpexDataClasses** record in both the ImportControl and the ExportControl objects. **ImpexDataClasses** has the following fields:

IDC_TEXT The data is in the Metafile Text format. This is identical to the Clipboard's CIF_TEXT format. In addition to text, this format records information about fonts, spacing, embedded graphics, etc.

IDC_GRAPHICS
The data is in the Metafile Graphics format. This is simply a GString in a VM chain. This format is identical to the Clipboard's CIF_GRAPHICS_STRING format.

IDC_SPREADSHEET
The data is in the Metafile Spreadsheet format. This corresponds to the Clipboard's CIF_SPREADSHEET format.

IDC_FONT
The data is in the Metafile Font format. This format is used to translate fonts between representations.

Every translation library translates a DOS file into a VM chain in one of these four formats, and vice versa. When an application declares an Impex object, it specifies what formats should be supported for that application. The user will be allowed to choose translation libraries which translate to or from a supported Metafile format.

16.2.1.2 ImpexTranslationParams

`ImpexTranslationParams`

The Import and Export objects have to pass information to the application and the translation libraries. The same sort of information gets passed in many situations; for example, the objects often have to pass the **VMFileHandle** and **VMBlockHandle** which specify the VM chain. For simplicity, the Impex objects just pass a pointer to a **ImpexTranslationParams** structure when they send messages to the translation libraries or the application. The library or application takes the appropriate action, changing the **ImpexTranslationParams** structure as necessary; it then sends a response message, which takes a pointer to the same **ImpexTranslationParams**. **ImpexTranslationParams** has the following structure:

```
typedef struct {
      optr              ITP_impexOD;
      Message           ITP_returnMsg;
      ImpexDataClasses  ITP_dataClass;
      VMFileHandle      ITP_transferVMFile;
      VMBlockHandle     ITP_transferVMChain;
      dword             ITP_internal;
      ManufacturerID    ITP_manufacturerID;
      ClipboardFormat   ITP_clipboardFormat;
} ImpexTranslationParams;
```

Objects

ITP_impexOD

This field holds the optr of whatever Impex object sent the message. The response message should be addressed to this optr.

ITP_returnMsg

This field holds the message which the library or application should send to the Impex object when it is finished. This message always takes a single argument, namely a pointer to the **ImpexTranslationParams**. The application should just pass the pointer to the **ImpexTranslationParams** to the routine **ImpexImportExportCompleted()**; this routine will send the appropriate notification message.

ITP_dataClass

This is a **ImpexDataClasses** record. Exactly one of the flags will be set, indicating what sort of Metafile format is being used.

ITP_transferVMFile

This is the **VMFileHandle** of the transfer file. The transfer file is automatically created and destroyed by an Impex object.

ITP_transferVMChain

This is the **VMBlockHandle** of the first block in the transfer VM chain. In Import operations, the translation library creates the VM chain in the transfer file; in Export operations, the application creates it. The chain will be freed when the Impex object destroys the transfer VM file. For information about creating VM chains, see section 18.4 of “Virtual Memory,” Chapter 18 of the Concepts Book.

ITP_internal

This field is for internal use by the Impex objects. You should not change it.

ITP_manufacturerID

This field contains the manufacturer ID which should be used for the Metafile data; see “The Clipboard,” Chapter 7 of the Concepts Book.

ITP_clipboardFormat

This field specifies what format should be used for the Metafile data; see “The Clipboard,” Chapter 7 of the Concepts Book.

The meaning of each field can change, depending on the circumstances. For example, when the ImportControl sends its message to the application, the *ITP_transferVMChain* field will contain the handle of a VM chain containing the imported data. In contrast, when the ExportControl sends its message, *ITP_transferVMChain* contains a null handle; the application should allocate a VM chain, fill it with data, and write the **VMBlockHandle** of the chain to this field. When a field has a special meaning, the documentation will explain it.

16.2.2 The ImportControl Object

Applications which use the Impex library will generally have a single ImportControl object. This object is usually a child of the File menu; however, applications may put it wherever they want. They should also place it on the application object's GAGCNLT_SELF_LOAD_OPTIONS General Change Notification list. Applications should disable this object (with MSG_GEN_SET_NOT_ENABLED) whenever they are not prepared to accept imported data; for example, some applications will choose to disable file importing whenever they are unable to open a new document.

The ImportControl object is subclassed from **GenControlClass** (by way of **ImportExportClass**, as noted above). It thus has all the functionality of that class. It also has a few instance data fields of its own; they are shown in Code Display 16-1.

Code Display 16-1 ImportControlClass Instance Data

```
/* ICI_attrs is a word-length record which stores attribute information for the
 * ImportControl object. It has only one flag, ICA_IGNORE_INPUT. */
    @instance ImportControlAttrs         ICI_attrs = 0;

/* ICI_dataClasses is a word-length record which indicates what Metafile formats
 * are supported by the application. The application must set this field. */
    @instance ImpexDataClasses           ICI_dataClasses = 0;
```

```
/* ICI_destination and ICI_message indicate what message should be sent when the
 * ImportControl object has finished importing a file. The application must set
 * these fields. The message must take a single argument, namely a pointer to an
 * ImpexTranslationParams structure in ss:bp (on the stack). */
    @instance optr                      ICI_destination;
    @instance word                      ICI_message;

/* Applications may wish to add their own UI objects to the Import dialog box.
 * They can do so by defining a generic tree (the top object of which must be not
 * “usable”), and putting an optr to the top object in ATTR_IMPORT_CONTROL_APP_UI.
 */
    @vardata optr                       ATTR_IMPORT_CONTROL_APP_UI;

/* Controller features flags */
typedef ByteFlags       ImportControlFeatures;
#define IMPORTCF_BASIC  0x01

typedef ByteFlags               ImportControlToolboxFeatures;
#define IMPORTCTF_DIALOG_BOX    0x01
```

16.2.2.1 ICI_attrs

ImportControlAttrs, MSG_IMPORT_CONTROL_GET_ATTRS, MSG_IMPORT_CONTROL_SET_ATTRS

ICI_attrs is a word-length record of type **ImportControlAttrs**. This record contains only one flag:

ICA_IGNORE_INPUT
If this flag is on, the ImportControl will consume all input to the application while the import occurs. By default, this flag is off.

To find out the current setting of this field, send MSG_IMPORT_CONTROL_GET_ATTRS to the ImportControl. To change this field, send MSG_IMPORT_CONTROL_SET_ATTRS to the ImportControl.

■ MSG_IMPORT_CONTROL_GET_ATTRS

```
ImportControlAttrs MSG_IMPORT_CONTROL_GET_ATTRS();
```

This message retrieves the current setting of the ImportControl's *ICI_attrs* field.

Source: Unrestricted.

Destination: Any ImportControl object.

Return: The ImportControl's *ICI_attrs* field.

Interception: This message should not be intercepted.

■ MSG_IMPORT_CONTROL_SET_ATTRS

```
void      MSG_IMPORT_CONTROL_SET_ATTRS(
          ImportControlAttrs  attrs);
```

This message changes the current settings of an ImportControl's *ICI_attrs* field.

Source: Unrestricted.

Destination: Any ImportControl object.

Parameters: *attrs* The new settings for the *ICI_attrs* field.

Interception: This message should not be intercepted.

16.2.2.2 ICI_dataClasses

```
MSG_IMPORT_CONTROL_GET_DATA_CLASSES,
MSG_IMPORT_CONTROL_SET_DATA_CLASSES
```

When you declare an Import object, you must specify what kind of Metafiles your application is prepared to accept. You do this by setting the value of the *ICI_dataClasses* field. This field is a word-length record of type **ImpexDataClasses** (described in section 16.2.1.2 on page 1026). If (for example) only the IDC_TEXT bit is set, the ImportControl will use only those import libraries which produce text Metafile output. More than one bit may be set; when the ImportControl sends its notification, it will tell the application what type of data is being imported.

To find out the current settings of the *ICI_dataClasses* field, send MSG_IMPORT_CONTROL_GET_DATA_CLASSES. To change the settings of this field, send MSG_IMPORT_CONTROL_SET_DATA_CLASSES.

■ MSG_IMPORT_CONTROL_GET_DATA_CLASSES

```
ImpexDataClasses MSG_IMPORT_CONTROL_GET_DATA_CLASSES();
```

This message retrieves the current setting of the ImportControl's *ICI_dataClasses* field. This tells you what kind of data can be imported.

Source: Unrestricted.

Destination: Any ImportControl object.

Return: The ImportControl's *ICI_dataClasses* field.

Interception: This message should not be intercepted.

■ MSG_IMPORT_CONTROL_SET_DATA_CLASSES

```
void        MSG_IMPORT_CONTROL_SET_DATA_CLASSES(
            ImpexDataClasses    dataClass);
```

This message changes the current settings of an ImportControl's *ICI_dataClasses* field.

Source: Unrestricted.

Destination: Any ImportControl object.

Parameters: *dataClass* The new settings for the *ICI_dataClasses* field.

Interception: This message should not be intercepted.

16.2.2.3 The ImportControl Action

```
MSG_IMPORT_CONTROL_GET_ACTION,
MSG_IMPORT_CONTROL_SET_ACTION,
ImpexImportExportCompleted()
```

The ImportControl does most of its work transparently to the application. It interacts with the rest of the application only when the user has selected a file to import and the appropriate translation library has produced a VM chain. At this point the ImportControl sends a notification message to the application. The application responds by copying the data from the VM chain and sending back an acknowledgment message. The ImportControl can then destroy the temporary VM transfer file.

The application determines what message will be sent, and to what object, by setting the *ICI_destination* and *ICI_message* fields. Whatever object will receive the message should define an appropriate message. The ImportControl will send this message with a single parameter: *itp*, a pointer to an **ImpexTranslationParams** structure (see section 16.2.1.2 on page 1026). The fields of the structure have the following meanings in this case:

ITP_impexOD
The object to which the application should send its acknowledgment message. In this case, it is the optr of the ImportControl.

ITP_returnMsg
The acknowledgment message to send when the import has been completed. In this case, it is MSG_IMPORT_CONTROL_IMPORT_COMPLETE.

ITP_dataClass
An **ImpexDataClasses** record with one flag set. This flag indicates what type of Metafile has been prepared.

ITP_transferVMFile
The **VMFileHandle** of the temporary transfer file.

ITP_transferVMChain
The **VMBlockHandle** of the lead block in the VM chain containing the imported data.

ITP_internal
For use by the ImportControl and should not be changed by the application.

The recipient of the message should take any appropriate action; usually this entails copying the data from the VM chain, as if it were pasting data from the Clipboard. When the application is finished, it should call **ImpexImportExportCompleted()**. This routine takes one parameter, namely the *itp* pointer which was passed to the object. (The **ImpexTranslationParams** structure should not have been changed.) **ImpexImportExportCompleted()** reads the appropriate message and destination from the **ImpexTranslationParams** and sends the proper acknowledgment message (which in this case is MSG_IMPORT_CONTROL_IMPORT_COMPLETE).

Applications which use the object model of document control will often set *ICI_destination* to TO_APP_TARGET; this will make it send its messages to the target object. The application can find out the ImportControl's action by sending it MSG_IMPORT_CONTROL_GET_ACTION. The application can change the ImportControl's action by sending it MSG_IMPORT_CONTROL_SET_ACTION.

■ MSG_IMPORT_CONTROL_GET_ACTION

```
void        MSG_IMPORT_CONTROL_GET_ACTION(
            ImpexAction *       retValue);
```

This message retrieves the values of an ImportControl's *ICI_destination* and *ICI_message* fields. These fields indicate what action the ImportControl will take when it is finished preparing a file for import.

Source: Unrestricted.

Destination: Any ImportControl object.

Parameters: *retValue* A pointer to an **ImpexAction** structure.

Return: The value of *ICI_message* (i.e. the message sent by the ImportControl).

recipient A pointer to an **ImpexAction** structure describing the message sent.

Interception: This message should not be intercepted.

Structures: The message and recipient are written to an **ImpexAction** structure:

```
typedef struct {
      word  message;    /* message sent */
      word  unused;
      optr  destOD;     /* Destination of message */
} ImpexAction;
```

■ MSG_IMPORT_CONTROL_SET_ACTION

```
void        MSG_IMPORT_CONTROL_SET_ACTION(
            optr   destOD,      /* Send messages to this object. */
            word   ICImsg);     /* Send this message to the above recipient. */
```

This message changes the values of an ImportControl's *ICI_destination* and *ICI_message* fields. These fields indicate what action the ImportControl will take when it is finished preparing a file for import.

Source: Unrestricted.

Destination: Any ImportControl object.

Parameters: *destOD* Set *ICI_destination* to this value.

ICImsg Set *ICI_message* to this value.

Interception: This message should not be intercepted.

16.2.2.4 Adding to the Import Dialog Box

When the user selects the "Import" trigger or tool, the Import controller brings up a dialog box. The application can, if it wishes, add UI objects to this box. It does so by defining a tree of generic objects (the top object of which must be set "not usable"). It must place an optr to the top object in the tree in ATTR_IMPORT_CONTROL_APP_UI. When the ImportControl builds the dialog box, it will add that optr as one of the children in the tree and set it "usable".

16.2.3 The ExportControl Object

Applications which use the Impex library will generally have a single ExportControl object. This object is usually a child of the File menu; however, applications may put it wherever they want. They should also place it on the application object's GAGCNLT_SELF_LOAD_OPTIONS GCN list. Applications should disable this object (with MSG_GEN_SET_NOT_USABLE) whenever they are not able to prepare data for export; for example, some applications will choose to disable file exporting whenever the "Cut" and "Copy" functions are disabled.

The ExportControl object is subclassed from **GenControlClass** (by way of **ImportExportClass**, as noted above). It thus has all the functionality of that class. It also has a few instance data fields of its own; they are shown in Code Display 16-2.

Code Display 16-2 ExportControlClass Instance Data

```
/* ECI_attrs is a word-length record which stores attribute information for the
 * ImportControl object. It has only one flag, ECA_IGNORE_INPUT. */
    @instance ExportControlAttrs         ECI_attrs = 0;

/* ECI_dataClasses is a word-length record which indicates what Metafile formats
 * are supported by the application. The application must set this field. */
    @instance ImpexDataClasses           ECI_dataClasses = 0;
```

```
/* ECI_destination and ECI_message indicate what message should be sent when the
 * ExportControl object is preparing to export a file. The application must set
 * these fields. The message must take a single argument, namely a pointer to an
 * ImpexTranslationParams structure in ss:bp (on the stack). */
    @instance optr                      ECI_destination;
    @instance word                      ECI_message;

/* Applications may wish to add their own UI objects to the Export dialog box.
 * They can do so by defining a generic tree (the top object of which must be not
 * "usable"), and putting an optr to the top object in ATTR_EXPORT_CONTROL_APP_UI.
 */
    @vardata optr                       ATTR_EXPORT_CONTROL_APP_UI;

/* Controller features flags */
typedef ByteFlags       ExportControlFeatures;
#define EXPORTCF_BASIC  0x01

typedef ByteFlags               ExportControlToolboxFeatures;
#define EXPORTCTF_DIALOG_BOX    0x01
```

16.2.3.1 ECI_attrs

ExportControlAttrs, MSG_EXPORT_CONTROL_GET_ATTRS, MSG_EXPORT_CONTROL_SET_ATTRS

ECI_attrs is a word-length record of type **ExportControlAttrs**. This record contains only one flag:

ECA_IGNORE_INPUT
If this flag is on, the ExportControl will consume all input to the application while the import occurs. By default, this flag is off.

To find out the current setting of this field, send MSG_EXPORT_CONTROL_GET_ATTRS to the ImportControl. To change this field, send MSG_EXPORT_CONTROL_SET_ATTRS to the ImportControl.

■ MSG_EXPORT_CONTROL_GET_ATTRS

```
ExportControlAttrs MSG_EXPORT_CONTROL_GET_ATTRS();
```

This message retrieves the current setting of the ExportControl's *ECI_attrs* field.

Source: Unrestricted.

Destination: Any ExportControl object.

Return: The ExportControl's *ECI_attrs* field.

Interception: This message should not be intercepted.

■ MSG_EXPORT_CONTROL_SET_ATTRS

```
void    MSG_EXPORT_CONTROL_SET_ATTRS(
        ExportControlAttrs  attrs);
```

This message changes the current settings of an ExportControl's *ECI_attrs* field.

Source: Unrestricted.

Destination: Any ExportControl object.

Parameters: *attrs* The new settings for the *ECI_attrs* field.

Interception: This message should not be intercepted.

16.2.3.2 ECI_dataClasses

MSG_EXPORT_CONTROL_GET_DATA_CLASSES,
MSG_EXPORT_CONTROL_SET_DATA_CLASSES

When you declare an Export object, you must specify what kind of Metafiles your application is able to create. You do this by setting the value of the *ECI_dataClasses* field. This field is a word-length record of type **ImpexDataClasses** (described in section 16.2.1.2 on page 1026). If (for example) only the IDC_TEXT bit is set, the ExportControl will use only those export libraries which expect text Metafile input. More than one bit may be set; when the ExportControl sends its notification, it will tell the application what type of data it expects to export.

To find out the current settings of the *ECI_dataClasses* field, send MSG_EXPORT_CONTROL_GET_DATA_CLASSES. To change the settings of this field, send MSG_EXPORT_CONTROL_SET_DATA_CLASSES.

■ MSG_EXPORT_CONTROL_GET_DATA_CLASSES

```
ImpexDataClasses MSG_EXPORT_CONTROL_GET_DATA_CLASSES();
```

This message retrieves the current setting of the ExportControl's *ECI_dataClasses* field. This tells you what kind of data can be exported.

Source: Unrestricted.

Destination: Any ExportControl object.

Return: The ExportControl's *ECI_dataClasses* field.

Interception: This message should not be intercepted.

■ MSG_EXPORT_CONTROL_SET_DATA_CLASSES

```
void      MSG_EXPORT_CONTROL_SET_DATA_CLASSES(
          ImpexDataClasses    dataClasses);
```

This message changes the current settings of an ExportControl's *ECI_dataClasses* field.

Source: Unrestricted.

Destination: Any ExportControl object.

Parameters: *dataClasses* The new settings for the *ECI_dataClasses* field.

Interception: This message should not be intercepted.

16.2.3.3 The ExportControl Action

MSG_EXPORT_CONTROL_GET_ACTION,
MSG_EXPORT_CONTROL_SET_ACTION

The ExportControl does most of its work transparently to the application. It interacts with the rest of the application after the user selects the name, location, and format of the exported file. At this point the ExportControl creates a temporary transfer file and sends a notification message to the application; the notification message passes the file handle and the format expected. The application responds by creating a VM chain in the transfer file and filling it with the data to export, formatted in the appropriate Metafile format. The ExportControl can then call the translation library to create the output file.

The application determines what notification message will be sent, and to what object, by setting the *ECI_destination* and *ECI_message* fields. Whatever object will receive the message should define an appropriate message. The ExportControl will send this message with a single parameter: *itp*, a pointer to an **ImpexTranslationParams** structure. The fields of the structure have the following meanings in this situation:

ITP_impexOD
The object to which the application should send its acknowledgment message. In this case, it is the optr of the ExportControl.

ITP_returnMsg
The acknowledgment message to send when the export has been completed. In this case, it is MSG_EXPORT_CONTROL_EXPORT_COMPLETE.

ITP_dataClass
An **ImpexDataClasses** record with one flag set. This flag indicates what type of Metafile should be prepared.

ITP_transferVMFile
The **VMFileHandle** of the temporary transfer file.

ITP_transferVMChain
A null handle. When the application has created the transfer VM chain, it should write the **VMBlockHandle** of the head of the chain to this field. If the application fails for any reason, it should leave this field as a null handle.

ITP_internal
This field is for use by the ExportControl and should not be changed by the application.

The recipient of the message should take any appropriate action; usually this entails translating the current selection into the Metafile format and writing it to a VM chain. When the application is finished, it should call **ImpexImportExportCompleted()**. This routine will send the appropriate acknowledgment message to the ExportControl object (in this case, MSG_EXPORT_CONTROL_EXPORT_COMPLETE). This routine takes one parameter, namely the *itp* pointer which was passed to the object. The *ITP_transferVMChain* field of the **ImpexTranslationParams** structure should be set to the handle of the head block in the VM chain. If the application was unable to prepare the data for export, it should clear this field.

The ExportControl object will have created a temporary file for the application to use. This file will be entirely empty when the application gets it. The ExportControl will ignore everything in the file except for the VM chain indicated by *ITP_transferVMChain*; thus, an application can feel free to allocate blocks in the VM file for scratch space. The ExportControl will destroy the file when the translation library has finished preparing the output file.

Applications which use the object model of document control will often set *ECI_destination* to TO_APP_TARGET; this will make it send its messages to the target object. The application can find out the ExportControl's action by sending it MSG_EXPORT_CONTROL_GET_ACTION. The application can change the ExportControl's action by sending it MSG_EXPORT_CONTROL_SET_ACTION.

■ MSG_EXPORT_CONTROL_GET_ACTION

```
void      MSG_EXPORT_CONTROL_GET_ACTION(
          ObjectState *       retValue);
```

This message retrieves the values of an ExportControl's *ECI_destination* and *ECI_message* fields. These fields indicate what action the ExportControl will take when it needs to have data prepared for export.

Source: Unrestricted.

Destination: Any ImportControl object.

Parameters: *retValue* A pointer to an **ObjectState** structure.

Return: The value of *ICI_message* (i.e. the message sent by the ExportControl).

recipient A pointer to an **ObjectState** structure describing the message sent.

Interception: This message should not be intercepted.

Structures: The message and recipient are written to an **ObjectState** structure:

```
typedef struct {
      int   notUsed;
      word  message;    /* Message sent */
      optr  destOD;     /* Destination of message */
} ObjectState;
```

■ MSG_EXPORT_CONTROL_SET_ACTION

```
void    MSG_EXPORT_CONTROL_SET_ACTION(
        optr   destOD,      /* Send messages to this object. */
        word   ECImsg);     /* Send this message to the above recipient. */
```

This message changes the values of an ExportControl's *ECI_destination* and *ECI_message* fields. These fields indicate what action the ExportControl will take when it is finished preparing a file for export.

16.3

Source: Unrestricted.

Destination: Any ExportControl object.

Parameters: *recipient* Set *ECI_destination* to this value.

message Set *ECI_message* to this value.

Interception: This message should not be intercepted.

16.2.3.4 Adding to the Import Dialog Box

When the user selects the "Export" trigger or tool, the Export controller brings up a dialog box. The application can, if it wishes, add UI objects to this box. It does so by defining a tree of generic objects (the top object of which must be set "not usable"). It must place an optr to the top object in the tree in ATTR_EXPORT_CONTROL_APP_UI. When the ExportControl builds the dialog box, it will add that optr as one of the children in the tree and set it "usable".

16.3 Writing Translation Libraries

GEOS comes with many translation libraries, and more are being added all the time. Geoworks is continually adding new translation libraries for popular formats. Nevertheless, we cannot guarantee to support every format. Developers may decide to write their own translation libraries. The current release of the GEOS SDK does not yet support developers writing their own translation libraries; however, we plan to support this by the final release of the SDK. This section describes how the libraries work in enough detail that developers will be able to do preliminary work in writing the libraries.

Advanced Topic

You do not need to read this section to use the Impex library.

Objects

Most applications will find the provided translation libraries sufficient for their needs. Therefore, most developers can skip this section. You should read this if you are planning on writing translation libraries, or just if you want more understanding of the importing and exporting process.

Remember: once a translation library has been written, all a user has to do to install it is copy it to the appropriate directory. All existing applications which use Impex will then automatically be able to import and export that format. Users will be able to buy translation library collections from third-party vendors the way they buy font collections now.

16.3.1 How Translation Libraries Work

A translation library's task is easily stated. It has to do two things: read a native-format file and produce a Metafile translation, and read data in a Metafile and write corresponding data in a native-format file. How easy this is to do depends on the formats involved.

Every translation library specifies what format of Metafile it expects to work with. For example, the FooWrite translation library would translate FooWrite files into IDC_TEXT Metafiles and vice versa. When a user activates the "Import" trigger, he will be presented with a list of formats to use; those formats will correspond to all the libraries which can translate files into formats the application can accept. For example, if the application specified that it could accept text or graphics Metafiles, the user's choice of format would depend on which of the installed libraries could generate text or graphics Metafiles.

The translation library can also suggest a file mask. For example, the FooWrite translation library might specify that FooWrite data files meet the pattern "*.FOO". By default, the Import file selector will show only the files that match the library's mask. However, the user can override this mask, setting a different one or no mask at all.

When a user decides to import a file, the ImportControl opens the source file and creates a temporary transfer VM file. The ImportControl then starts up the appropriate translation library and passes the two file handles to it. The translation library should read the entire source file, translate it to the appropriate Metafile format, and write it to a VM chain in the transfer file. It then returns the **VMBlockHandle** of the head of the **VMChain** to the ImportControl. If it was unable to translate the file, it should return a null handle; the ImportControl then displays an appropriate error message. The ImportControl will close the source file automatically.

16.3

When a user decides to export a file, the ExportControl creates a temporary transfer file and opens an empty native-format file for the output. The ExportControl then calls the translation library. The library is passed the handles of the two files, as well as the handle of the Metafile VM chain. The library reads the Metafile and writes an appropriate data file. When it is finished, it notifies the ExportControl, which automatically closes the destination file and destroys the temporary transfer file.

16.3.2 Intermediate Formats

Many libraries will want to make use of intermediate formats. For example, a company may have defined its own transfer format for its applications. The simplest way for it to translate files into the GEOS Metafile might be to translate the file into its own transfer format, then translate from this format into the Metafile format. GEOS supports this with its use of intermediate translation libraries.

An intermediate library is much like an ordinary translation library. Like other libraries, it must translate from its own format into the Metafile format. The only difference is that intermediate libraries are not called by the Impex objects; instead, they are called by other translation libraries.

For example, suppose FooWare, Inc., has a line of graphic FooApps which includes FooDraw, FooPaint, FooSketch, and FooScribble. FooWare has developed its own file-transfer format, FooInterchangeFormat (FIF); it has code written to translate any FooApp's files into FIF and back.

FooWare now wants to write translation libraries for GEOS. The first thing FooWare does is write an intermediate translation library which takes a DOS file containing FIF data and produces a VM chain containing a GEOS graphic Metafile, and vice versa. Once this is written, FooWare has an easy time writing the actual translation libraries. For example, the FooPaint translation library imports files by converting a FooPaint data file into the analogous FIF file. Since FooWare already has routines to do this translation, it just has to port existing code to the GEOS library. The FooPaint translation library then calls the FIF-to-Metafile intermediate translation library and gets the finished translation. Similarly, the FooPaint translation library exports data by calling the FIF-to-Metafile library to produce a FIF version of the data; it can then use ported code to produce an actual FooPaint file.

Note that any ordinary translation library can also be used as an intermediate translation library. For example, let's suppose that FooWare has code written to translate FIF files into PostScript files. This makes translating the documents even easier. The FIF-to-Metafile library can just use ported code to produce a PostScript version of the data; it can then call the PostScript-to-Metafile translation library, which is provided with GEOS. GEOS comes with translation libraries for many popular file-interchange formats; thus, many developers will be able to write translation libraries just by porting code from their pre-existing translators, then calling one of the GEOS translation libraries.

16.3

◆Objects

The Spool Library

17

Any geode can work with printers, fax machines, and other print devices by including the spool library and instantiating a PrintControl object to handle system interactions. The **PrintControlClass** provides the printing interface. It includes both user interface and an interface between the geode and the printing thread. There is also a **PageSizeControlClass** which provides UI concerning page layout.

17.1

Before reading this chapter, you should be familiar with the graphics system.

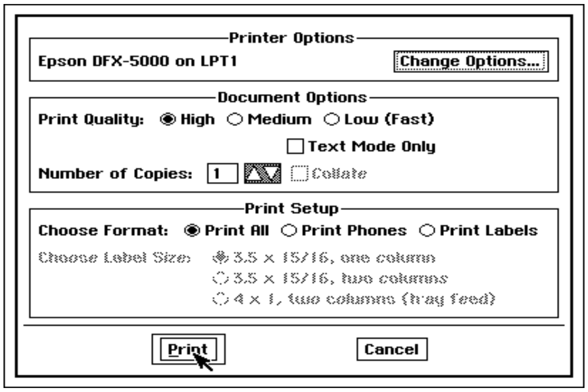
17.1 Introduction to Printing

By including a PrintControl object, an application gains access to the powers of the GEOS printing system. Below are several of the features built into GEOS:

- Single Imaging Model
  You'll use the same commands to describe print jobs as you did for screen drawings. These commands work for all supported printers. Thus, anything you can draw can be printed, and the application doesn't have to worry about what model of printer is being used.

- Background Printing
  Background printing allows users to continue working while their documents print. This is an example of multitasking at work. A process known as the Spooler spawns a separate thread for each active printer. Since each printer's thread priority is low, printing will take place in the background, interfering little with the user's interactions. If you want printing to take place in the foreground, your application can boost the priority of a printer thread.

- Standard UI Components
  The PrintControl is a full-featured control object and it provides a UI mechanism for finding out what a user wants to print. For user choices which depend on the printer (for example, some printers have no low-quality mode), the PrintControl will take printer abilities into consideration. (See Figure 17-1.)



Printer Options

Epson DFX-5000 on LPT1

Document Options

Print Quality: ● High ○ Medium ○ Low (Fast)

□ Text Mode Only

Number of Copies: 1 □ Collate

Print Setup

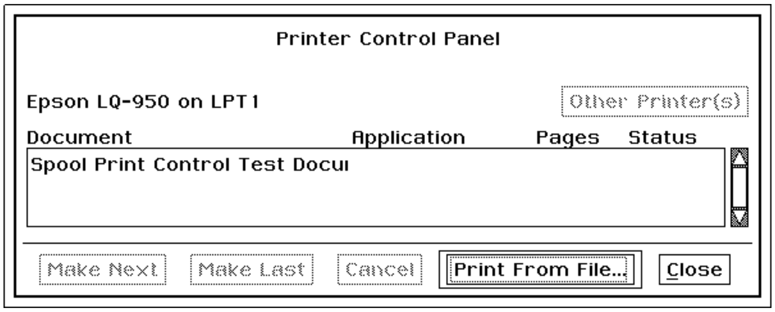
Choose Format: ● Print All ○ Print Phones ○ Print Labels

Choose Label Size: ● 3.5 x 15/16, one column ○ 3.5 x 15/16, two columns ○ 4 x 1, two columns (tray feed)

Figure 17-1 *Print Dialog Box*

This Print dialog box shows a Print Control plus some specialized gadgets supplied by an application.

- Control Over Scheduling
  The Spooler normally handles jobs in FIFO (first in, first out) order. Geodes can exercise a great deal of control over the Spooler's scheduling of jobs. Any scheduling options the user might request using the Printer Control Panel (see Figure 17-1), the application can request using spool routines.



Printer Control Panel

Epson LQ-950 on LPT1

Document	Application	Pages	Status
Spool Print Control Test Docu			

Figure 17-2 *Printer Control Panel*

- Managing the Printer Yourself
  If you can't or won't use standard GEOS commands to describe a print job, you can use Raw Mode to send a packet of commands in the printer's own language. For example, you could send escape codes or commands from a

page description language. Since this is a standard printing mode, you still benefit from all the usual spool scheduling features.

- Management of Multiple Printers
  The user may have multiple printing devices hooked up and installed at one time. The system handles this situation intelligently: the Spooler maintains a separate queue for each serial or parallel port. Thus, if two printers are installed on two different ports, both may print at once. If two or more printers are installed on the same port then they share a queue (see Figure 17-3). Thus, jobs for one printer won't try to print at the same time as those for another on the same port. To make sure the job prints to the correct printer, the Spooler will put up a dialog box advising the user to put the correct device on line.

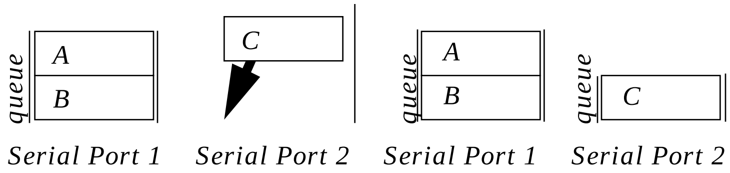


Figure 17-3 *Multiple Printer Queues*

The Spooler maintains separate queues for separate ports. If all printers shared the same queue, the incoming job C wouldn't be able to print until after jobs A and B had finished. Instead, job C will be the next thing printed on its device.

17.2 Simple Printing Example

Adding printing capability to an application is fairly simple, especially if it will be printing WYSIWYG. Somewhere in the application there is probably a message handler which is in charge of drawing the document. By including a Print Control object and writing handlers to some standard messages, it is a simple manner to redirect the document drawing commands to a printer.

Code Display 17-1 Printing Example

```
@object GenApplicationClass MyApp = {
        /* …Much other GenApplication instance data omitted… */
        gcnList(MANUFACTURER_ID_GEOWORKS, GAGCNLT_SELF_LOAD_OPTIONS) =
                @MyPrintControl;
        ATTR_GEN_APPLICATION_PRINT_CONTROL = @MyPrintControl;
}

@object PrintControlClass MyPrintControl = {
/* The PrintControlClass is a subclass of GenControlClass. Its “features”
 * correspond to the “Print” and “Fax” triggers in the File menu. Its instance
 * data contains information about the document to be printed, and links to
 * objects which will provide information. */

/* Single page documents would leave out page range UI from the Print dialog:
        PCI_attrs =     (@default & ~(PCA_PAGE_CONTROLS| PCA_VERIFY_PRINT));
        PCI_startUserPage = 1;
        PCI_endUserPage = 1 */

/* The PCI_output, or Print Output object, is in charge of supplying the graphics
 * commands describing the print job whenever the user wants to print. This object
 * is expected to handle a MSG_PRINT_CONTROL_START_PRINTING. */

 * The print output object is normally either the model, target or process. */
        PCI_output =    TO_APP_TARGET;

/* The PCI_docNameOutput object is supposed to provide the name of the document
 * on demand. This name is used to identify the document to the user in the Print
 * Control Panel. If this object is a GenDocumentGroup, it automatically does this.
 * If the object in the following field is not a GenDocumentGroup object, then
 * it must have a handler for MSG_PRINT_GET_DOC_NAME. */
        PCI_docNameOutput = MyGenDocumentGroup;

/* Many simple applications only support one document size,
 * and may specify it in the PrintControl and never bother with it again: */
        PCI_docSizeInfo = {     (15/2*72),
                                (19/2*72),
                                0,
                                {0, 0, 0, 0}}; */
/* Many simple applications do not support multiple documents. These applications
 * should set the GS_ENABLED flag of the PrintControl’s GI_states field. */
}
```

◆Objects

```
@method MVTStartPrinting, MyVisTargetedClass, MSG_PRINT_START_PRINTING
/* We’ve set up our Print Control to send this message to the Target, so whatever
 * object will have the target when the user wants to print needs a handler for
 * this message. We could have easily have put the Model or process in charge
 * of printing, changing PCI_output accordingly. It is often convenient to use
 * the same object to handle drawing to screen and to the printer. */

/* Arguments:   optr            printControlOD,
                GStateHandle    gstate */

{
        PCMarginParams margins;

/*
 *      Applications which allow the user to change the document size may handle
 *      the situation in more than one way. If the user has changed the page setup
 *      by working with a PageSizeControl, then the application has probably
 *      already been alerted to the page size. If the application has its own
 *      way of computing document size, it should use it. Such applications should
 *      send MSG_PRINT_CONTROL_SET_DOC_SIZE and MSG_PRINT_CONTROL_SET_DOC_MARGINS
 *      to the PrintControl, either in this handler or else whenever the page size
 *      changes. Either time is fine, just so long as the document size is set
 *      correctly by the time this MSG_PRINT_START_PRINTING handler is finished.

 *      Applications supporting only a single document size should probably set
 *      the size (and margins) in the PrintControl’s instance data, as shown
 *      above. It is also possible to send a MSG_PRINT_CONTROL_SET_DOC_SIZE on
 *      every print, but if the size never changes, this is a bit wasteful.

 *      This application supports only one size of document. If the document
 *      doesn’t fit on the page, the spooler will tile it onto multiple pages
 *      as necessary. It uses the printer’s margins as the document margins: */

        @call   MyPrintControl::MSG_PRINT_CONTROL_GET_PRINTER_MARGINS(
                &margins,       /* Fill in structure with printer’s margins */
                TRUE);          /* ...and automatically use printer margins
                                 * as document margins */

        @call   self::MSG_VIS_DRAW(DF_PRINT, gstate);

        /* If the MSG_VIS_DRAW handler didn’t end with a form feed, put one in:
        GrNewPage(gstate, PEC_FORM_FEED); */

        @send   MyPrintControl::MSG_PRINT_CONTROL_PRINTING_COMPLETED();
}
```

```
@method MVTDraw, MyVisTargetClass, MSG_VIS_DRAW
/* Chances are, whatever object this is has some sort of draw handler already.
 * However, if you’re building up this class from scratch, you’ll be glad to
 * know that commands of the following form work just as well drawing in
 * response to a print request as they do drawing anything else: */

{       GrSetLineColor(gstate, CF_INDEX, C_RED, 0, 0);
        GrDrawLine(gstate, 144, 144, 288, 288);
        GrNewPage(gstate, PEC_FORM_FEED);
}
```

17.3 How Jobs Get Printed

You may wonder how and when these various features get implemented. Simply stated, the application describes the print job, which gets placed onto a queue where it remains until it is fed to a printer. For many programmers, that’s enough to know. If you want a more complete explanation, read on.

17.3.1 Printing System Components

Printing involves the concerted effort of several objects running in different threads. The most important pieces of the printing system are

PrintControl

The printing system is big and would be difficult to interact with if not for the Print Control, which acts as a sort of intermediary. This object is a member of the **PrintControlClass**, a GenControl subclass provided by the spool library. It handles the UI, communication with the spool thread, and most other printing functions. Every geode which is going to print includes its own Print Control, placing the control object in its generic tree.

Print Output

The Print Output is an object (often the process object) chosen by the geode when creating the Print Control. This object is in charge of building print jobs and must be prepared to provide page descriptions whenever it receives a MSG_PRINT_START_PRINTING from the Print Control.

Spooler The Spooler handles most of the important “behind the scenes” work of printing. As mentioned, it handles the scheduling of jobs and manages the operations by which generic print jobs are translated for specific printers.

Printer Driver
GEOS printer drivers handle the back end of printing. GEOS has many printer drivers, each of which serves one or more models of printer. The driver supplies the Spooler with information about the printer and makes some final adjustments to the translated print job to get it ready for the specific printer.

17.3.2 Chronology

Now that you’re somewhat familiar with these components, you’re ready for the detailed account of how they work together to print a job. Note that the application is only involved at one point. The printing system takes care of the rest automatically.

1 **User Activates Trigger**
Printing normally begins when the user activates the User Print Trigger provided by the Print Control (see Figure 17-4). This trigger sends a message to the Print Control telling it to put up the Print Dialog box.

2 **User Interacts with the Print Dialog Box**
The Print Control responds to the Print Trigger’s message by presenting a dialog box in which the user selects such things as page range and print quality (see Figure 17-1 on page ◆ 1048). The user then either initiates printing or cancels. If the user wants to print, he’ll confirm this by clicking on the appropriate trigger. The application will then get a chance to veto the user’s choices. The Print Control asks the dialog box to remove itself, and it then gets ready for the upcoming print job.

3 **Print Control Prepares for New Job**
Soon the Print Control is going to ask for the page descriptions, but first it needs a place to store the data. It creates a spool file in which to store the upcoming graphics commands. It also allocates a GString handle by which the Print Output can transmit the commands. It then sends a

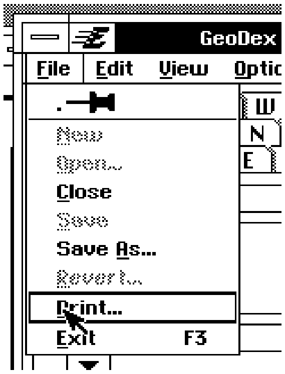


Figure 17-4 *User Print Trigger*

In some specific user interfaces, the standard User Print Trigger is an entry in the File Menu.

message to the application's Print Output, signalling that the Print Output should supply the job. (See Figure 17-5.)

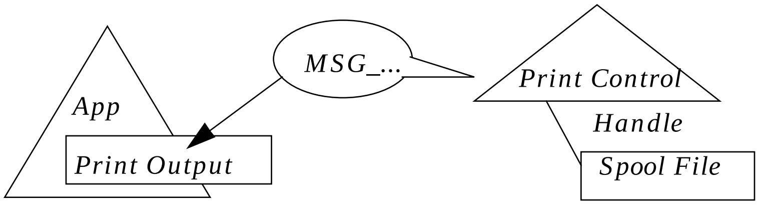


Figure 17-5 *Preparing for New Print Job*

The Print Control readies a file to hold data and alerts the application

4 **Application Supplies Job**
Having received the Print Control's message, the Print Output supplies graphics routines to the provided GString handle, using the same commands as when printing to any other GState. This string of graphics commands ends with a message saying the job is completed. The Print Control supplies some extra information to the spooler, and the print job is ready to enter the queue. (See Figure 17-6.)

5 **Job Traverses Queue**
At this point, the spool library places the print job on a printer queue where the job waits to be printed. (Actually, the Spooler uses a trick to

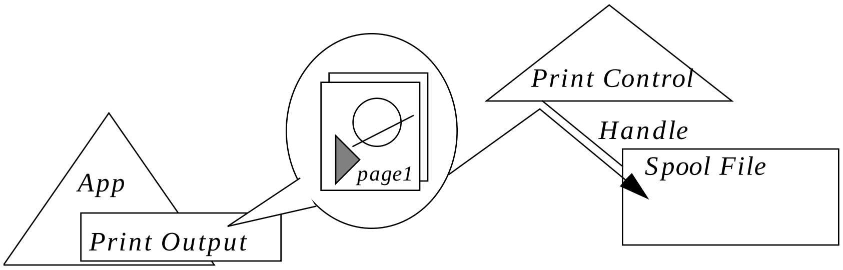


Figure 17-6 *Application Supplies Job*

The application supplies a page description, which goes through the GString handle supplied by the Print Control into the Spool File.

save on overhead. If there are no jobs for a printer, the Spooler doesn't maintain a queue for that printer. When a new job comes in for a printer that has no queue so far, a new queue is prepared for that printer and the job placed on the new queue. When the last job of a queue is printed and there are no more jobs forthcoming, the queue is removed.) The Spooler's various scheduling powers, if exercised, affect jobs in the queue. (See Figure 17-7.)

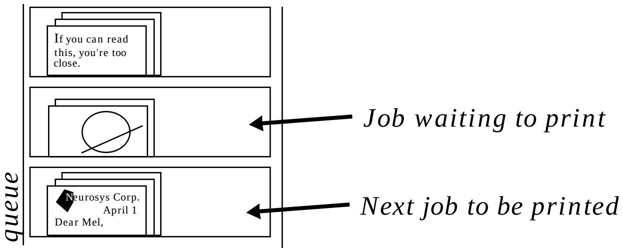


Figure 17-7 *Job Traverses Queue*

6 **Printing**
Eventually, the job reaches the head of the queue. It is now ready to print. The Spooler works with the Printer Driver to transform the print job into a form that the printer can work with. This may involve building a bitmap depicting the job, or it might involve translating the GString's commands into the commands of some other page description language, such as PostScript. (See Figure 17-8.)

7 **Hard Copy**
The printer driver sends the appropriate printer commands to the

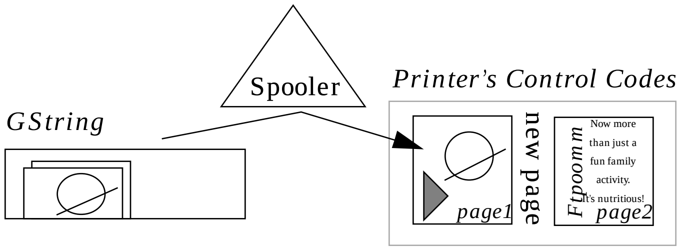


Figure 17-8 *Printing*

The Spooler converts the print job to a format the printer can understand.

printer. If the printer doesn’t have sufficient memory to hold the whole job, the driver sends only part of the job at a time.

17.4 Print Control Instance Data

The PrintControl is a powerful and adaptable subclass of GenControl and contains many instance fields to fit the needs of different types of applications. Each field is described in greater detail later in this section; all are shown in Code Display 17-2.

Code Display 17-2 Print Control Instance Data and Features

```
        /* The following bitfield contains the PrintControl’s attributes */
@instance PrintControlAttrs PCI_attrs =
                        (PCA_COPY_CONTROLS | PCA_PAGE_CONTROLS |
                         PCA_QUALITY_CONTROLS | PCA_USES_DIALOG_BOX |
                         PCA_GRAPHICS_MODE | PCA_TEXT_MODE );
        /* Possible PCI_attrs flags (may be combined using | and &):
         * PCA_MARK_APP_BUSY,           PCA_VERIFY_PRINT,
         * PCA_SHOW_PROGRESS,           PCA_PROGRESS_PERCENT,
         * PCA_PROGRESS_PAGE,           PCA_FORCE_ROTATION
         * PCA_COPY_CONTROLS,           PCA_PAGE_CONTROLS,
         * PCA_QUALITY_CONTROLS,        PCA_USES_DIALOG_BOX,
         * PCA_GRAPHICS_MODE,           PCA_TEXT_MODE
         * PCA_DEFAULT_QUALITY                           */
        /* The fields below are the complete and requested page ranges */
@instance word  PCI_startPage = 1;
@instance word  PCI_endPage = 1;
@instance word  PCI_startUserPage = 0;
@instance word  PCI_endUserPage = 0x7fff”;

        /* The following field contains the default printer number */
@instance word  PCI_defPrinter = -1;

        /* The following fields deal with document dimensions */
@instance PageSizeReport PCI_docSizeInfo = 0;

        /* Pointers to objects receiving vital messages */
@instance optr  PCI_output;
@instance optr  PCI_docNameOutput;

        /* The PrintControl’s features determine whether to use standard
         * print and fax triggers to bring up the Print Dialog box. */
typedef ByteFlags PrintControlFeatures;
/* The following flags may be combined with | and &:
        PRINTCF_PRINT_TRIGGER,
        PRINTCF_FAX_TRIGGER */
/* To provide a non-standard print trigger, use the GenControl vardata
 * field ATTR_GEN_CONTROL_APP_UI. */
typedef ByteFlags PrintControlToolboxFeatures;
/* The following flags may be combined with | and &:
        PRINTCTF_PRINT_TRIGGER
        PRINTCTF_FAX_TRIGGER */

/* To include application-specific UI in the print dialog box. */
@vardata optr ATTR_PRINT_CONTROL_APP_UI;

/* This piece of temporary vardata is internal: */
@vardata TempPrintCtrlInstance TEMP_PRINT_CONTROL_INSTANCE;

/* Its structures are defined as follows:
typedef struct {
 optr                   TPCI_currentSummons; ( currently active summons )
 optr                   TPCI_progressBox; ( OD of progress dialog box */
 ChunkHandle            TPCI_jobParHandle; ( handle to JobParamters )
 word                   TPCI_fileHandle; ( file handle (if printing) )
 word                   TPCI_gstringHandle; ( gstring handle if printing )
 word                   TPCI_printBlockHan; ( the printer block handle )
 PrintControlAttrs      TPCI_attrs;
 PrintStatusFlags       TPCI_status;
 byte                   TPCI_holdUpCompletionCount;
} TempPrintCtrlInstance;
typedef ByteFlags PrintStatusFlags;
```

```
#define PSF_FAX_AVAILABLE 0x80 ( set if a fax driver is available )
#define PSF_ABORT 0x08 ( user wants to abort printing )
#define PSF_RECEIVED_COMPLETED 0x04 ( MSG_…_PRINTING_COMPLETED received )
#define PSF_RECEIVED_NAME 0x02 ( MSG_PC_SET_DOC_NAME received )
#define PSF_VERIFIED 0x01 ( PSG_PC_VERIFY_? received ) */

@vardata TempPrintCompletionEventData TEMP_PRINT_COMPLETION_EVENT;

/* The TempPrintCompletionEventData structure is defined:
        typedef struct {
         MemHandle      TPCED_event;
         MessageFlags   TPCED_messageFlags;
        } TempPrintCompletionEventData;
```

17.4.1 Alerting the GenApplication

PrintControl objects should be placed on the GenApplication's MANUFACTURER_ID_GEOWORKS/GAGCNLT_SELF_LOAD_OPTIONS general change notification list. Also, if the application is to allow printing of documents from the file from the file manager, then the PrintControl's optr should be specified in the GenApplication's ATTR_GEN_APPLICATION_PRINT_CONTROL field.

17.4.2 Attributes

PCI_attrs, MSG_PRINT_CONTROL_SET_ATTRS, MSG_PRINT_CONTROL_GET_ATTRS

The Print Control includes several attributes which are grouped together into a record of type **PrintControlAttrs**. These represent some choices made when instantiating the Print Control.

PCA_MARK_APP_BUSY

Describing large print jobs can take a fair amount of time. If this bit is set, the application will reassure the user that it is busy (by showing the busy cursor) while spooling long print jobs.

PCA_VERIFY_PRINT

If this bit is set, the Print Control will ask the application for confirmation before printing anything. This request will come in the form of a MSG_PRINT_VERIFY_PRINT_REQUEST sent to the *PCI_output* object. The Print Control sends this message after the user has finished interacting with the Print dialog box so that the application can make sure that the user has made valid choices.

PCA_SHOW_PROGRESS, PCA_PROGRESS_PERCENT, PCA_PROGRESS_PAGE 17.4

When describing a large print job, it is considerate to let the user know when the application is making some kind of progress. If the first of these bits is set, the Print Control will display a progress dialog box when spooling a print job. The next two bits determine whether progress will be reported as a percentage, number of pages completed, both, or neither. Regardless of whether percents or page number progress is reported, the progress box can display an application-supplied string.

PCA_FORCE_ROTATION

If this bit is set, the output will automatically be printed rotated on the page. Normally, this bit is not set so the Spooler will make a choice to make the job fit on the minimum number of pages. The Banner program sets this bit to make sure that fanfold paper printers will create continuous banners; if it didn't print sideways, then users would have to use a lot of tape to put their banners together (they have to anyhow, if using non continuous-feed paper).

PCA_COPY_CONTROLS

This flag determines whether the Print Control will allow the user to request that multiple copies be printed. If your application uses this option, then it is vital that the print control be updated whenever the total page count of the document changes.

PCA_PAGE_CONTROLS

If this bit is set, the user will be allowed to select specific page ranges to print. Even if it seems like users would have little reason to print short ranges, it's still nice to include this option: if the printer messes up one page, it's usually irritating to have to reprint the whole document.

PCA_QUALITY_CONTROLS

If this flag is on, the user will be allowed to select which quality to print with. Note that not all printers can print at all qualities, but most support more than one mode.

PCA_USES_DIALOG_BOX

This bit determines whether the Print Control will display a dialog box containing printing choices for the user. Specialty geodes and GCM applications might want to turn this flag off; it is set by default. 17.4

PCA_GRAPHICS_MODE, PCA_TEXT_MODE

These bits determine whether the Control should support graphic mode and/or text mode output.

PCA_DEFAULT_QUALITY

These two bits contain the default print quality to use. The enumerated type **PrintQualityEnum** defines the values available: PQT_HIGH, PQT_MEDIUM, and PQT_LOW.

Those attributes which are on by default are PCA_COPY_CONTROLS, PCA_PAGE_CONTROLS, PCA_QUALITY_CONTROLS, PCA_USES_DIALOG_BOX, PCA_GRAPHICS_OUTPUT, and PCA_TEXT_OUTPUT.

There are messages to get and set these attributes.

■ MSG_PRINT_CONTROL_SET_ATTRS

```
void        MSG_PRINT_CONTROL_SET_ATTRS(
            PrintControlAttrs attributes);
```

Use this message to change the values stored in the *PCI_attrs* structure. Pass a record of type **PrintControlAttrs** containing the desired values.

Source: Unrestricted.

Destination: Any PrintControl object.

Parameters: *attributes* The new **PrintControlAttrs**.

Return: Nothing.

Interception: Unlikely.

◆ Objects

■ MSG_PRINT_CONTROL_GET_ATTRS

```
PrintControlAttrs MSG_PRINT_CONTROL_GET_ATTRS();
```

Use this message to retrieve the values stored in the *PCI_attrs* structure. It will return a record of type **PrintControlAttrs** containing the desired values.

Source: Unrestricted.

Destination: Any PrintControl object.

Parameters: None.

Return: The present print control attributes.

Interception: Unlikely.

17.4.3 Page Range Information

```
PCI_startPage, PCI_endPage, PCI_startUserPage,
PCI_endUserPage, MSG_PRINT_CONTROL_SET_TOTAL_PAGE_RANGE,
MSG_PRINT_CONTROL_GET_TOTAL_PAGE_RANGE,
MSG_PRINT_CONTROL_SET_SELECTED_PAGE_RANGE,
MSG_PRINT_CONTROL_GET_SELECTED_PAGE_RANGE
```

The *PCI_startPage* and *PCI_endPage* fields should contain the page numbers of the first and last pages possible to print, known as the total page range. The *PCI_startUserPage* and *PCI_endUserPage* contain the beginning and ending page numbers of the range the user wants to print, often called the user page range.

For documents whose page length may change (such as word processor documents), the page ranges should be updated whenever the Print dialog box goes up, at which time the application will be receiving a MSG_PRINT_NOTIFY_PRINT_DB. Normally, the user page range will start out being the same as the total page range. For most documents the first page of the total page range is page one, but applications which will include cover sheets might want the cover sheet to be page zero.

■ MSG_PRINT_CONTROL_SET_TOTAL_PAGE_RANGE

```
void      MSG_PRINT_CONTROL_SET_TOTAL_PAGE_RANGE(
          int    firstPage,
          int    lastPage);
```

The application should send this message to set the first and last page numbers of the document. You might want to send this message every time the document length changes or just every time the Print dialog box is put up. The Print Control warns the *PCI_output* object of the occurrence of the latter event with a MSG_PRINT_NOTIFY_PRINT_DB.

Source: Unrestricted.

Destination: Any PrintControl object.

Parameters:

firstPage	The page number of the document's first page.
lastPage	The page number of the document's last page.

Return: Nothing.

Interception: Unlikely.

■ MSG_PRINT_CONTROL_GET_TOTAL_PAGE_RANGE

```
dword     MSG_PRINT_CONTROL_GET_TOTAL_PAGE_RANGE();
```

This message returns two integers. These integers are the numbers of the first and last pages of the range of possible pages. These values aren't necessarily the first and last pages of the range the user wants to print. Use MSG_PRINT_CONTROL_GET_SELECTED_PAGE_RANGE for that information.

Source: Unrestricted.

Destination: Any PrintControl object.

Parameters: None.

Return: A double word. The high word is the first page; the low word is the last page.

Interception: Unlikely.

◆ Objects

■ MSG_PRINT_CONTROL_SET_SELECTED_PAGE_RANGE

```
void      MSG_PRINT_CONTROL_SET_SELECTED_PAGE_RANGE(
          int    firstPage,
          int    lastPage);
```

This message takes the passed values and uses them as the first and last pages that the user wants to print.

Source: Unrestricted.

Destination: Any PrintControl object.

Parameters: *firstPage* The page number of the first page to print.

lastPage The page number of the last page to print.

Return: Nothing.

Interception: Unlikely.

■ MSG_PRINT_CONTROL_GET_SELECTED_PAGE_RANGE

```
dword     MSG_PRINT_CONTROL_GET_SELECTED_PAGE_RANGE();
```

This message returns the user's selected range of pages to print.

Source: Unrestricted.

Destination: Any PrintControl object.

Parameters: None.

Return: A double word. The high word is the first page which the user wishes to print; the low word is the last page of this range.

Interception: Unlikely.

17.4.4 Document Size

```
PCI_docSizeInfo, MSG_PRINT_CONTROL_SET_DOC_SIZE,
MSG_PRINT_CONTROL_GET_DOC_SIZE,
MSG_PRINT_CONTROL_SET_DOC_MARGINS,
MSG_PRINT_CONTROL_GET_DOC_MARGINS,
MSG_PRINT_CONTROL_SET_EXTENDED_DOC_SIZE,
MSG_PRINT_CONTROL_GET_EXTENDED_DOC_SIZE,
```

`MSG_PRINT_CONTROL_SET_DOC_SIZE_INFO,`
`MSG_PRINT_CONTROL_GET_DOC_SIZE_INFO`

It is possible to specify the size of the document when creating the Print Control. Note that this is the size of the document, not the size of the piece of paper. Ideally, the document should fit on the paper, though obviously in the case of huge documents like some spreadsheets, this may not be the case. The document size includes the margin size; it is possible to set the margin size as well.

The document size must be set correctly before the document is finished printing. If all documents the application produces have the same dimensions (or if you want some size to be the default), you may specify dimensions for the document when instantiating the print control. You may also set up document margins at this time.

The messages listed above get and set the document and margin sizes. You must use MSG_PRINT_CONTROL_GET_EXTENDED_DOC_SIZE and MSG_PRINT_CONTROL_SET_EXTENDED_DOC_SIZE when working with the dimensions of 32–bit extended documents.

■ MSG_PRINT_CONTROL_SET_DOC_SIZE

```
void      MSG_PRINT_CONTROL_SET_DOC_SIZE(
          int    width,
          int    height);
```

This message changes the values of the document size. It takes two integers, representing the new width and height for the document to use. It can only use 16-bit values, so use MSG_PRINT_CONTROL_SET_EXTENDED_DOC_SIZE when working with 32-bit extended graphics spaces.

Remember that the document size includes the document margins.

Source: Unrestricted.

Destination: Any PrintControl object.

Parameters: *width* The document's width, in points.

height The document's height, in points.

Return: Nothing.

Interception: Unlikely.

■ MSG_PRINT_CONTROL_GET_DOC_SIZE

```
dword    MSG_PRINT_CONTROL_GET_DOC_SIZE();
```

Use this message to retrieve the present document size. The size is returned as a width and height, each expressed in points.

Note that if the size might be a 32-bit value (which might happen if the document uses an extended graphics space), you must use the MSG_PRINT_CONTROL_GET_EXTENDED_DOC_SIZE. If either dimension is a 32-bit number, using a regular MSG_PRINT_CONTROL_GET_DOC_SIZE will result in an error.

Remember that the document size includes the document margins.

Source: Unrestricted.

Destination: Any PrintControl object.

Parameters: None.

Return: A double word. The high word is the width, the low word is the height.

Interception: Unlikely.

■ MSG_PRINT_CONTROL_SET_EXTENDED_DOC_SIZE

```
void     MSG_PRINT_CONTROL_SET_EXTENDED_DOC_SIZE(
         PCDocSizeParams *ptr);
```

This message changes the values of the document size using the two passed double integers as the new width and height to use. When working with normal 16 bit graphics spaces, use MSG_PRINT_CONTROL_SET_DOC_SIZE instead.

Remember that the document size includes the document margins.

Source: Unrestricted.

Destination: Any PrintControl object.

Parameters: *ptr* Pointer to a **PCDocSizeParams** structure containing the document size.

Return: Nothing.

Interception: Unlikely.

Structures: The **PCDocSizeParams** structure has the following definition:

```
typedef struct {
      dword       PCDSP_width;
      dword       PCDSP_height;
} PCDocSizeParams;
```

■ MSG_PRINT_CONTROL_GET_EXTENDED_DOC_SIZE

```
void    MSG_PRINT_CONTROL_GET_EXTENDED_DOC_SIZE(
        PCDocSizeParams *ptr);
```

Use this message to retrieve the present document size. It returns two double integers representing the width and height, expressed in points. If the size is a 16 bit value, you can use MSG_PRINT_CONTROL_GET_DOC_SIZE instead.

Remember that the document size includes the document margins.

Source: Unrestricted.

Destination: Any PrintControl object.

Parameters: *ptr* Pointer to a **PCDocSizeParams** structure to hold document parameters.

Return: Nothing is returned explicitly.

ptr The structure is filled with document size.

Interception: Unlikely.

Structures: The **PCDocSizeParams** structure has the following definition:

```
typedef struct {
      dword       PCDSP_width;
      dword       PCDSP_height;
} PCDocSizeParams;
```

■ MSG_PRINT_CONTROL_SET_DOC_MARGINS

```
void    MSG_PRINT_CONTROL_SET_DOC_MARGINS(
        PCMarginParams *ptr);
```

Use this message to set new values for the document margins. It takes four arguments, the point values to use for the left, top, right, and bottom margins.

Source: Unrestricted.

Destination: Any PrintControl object.

Parameters: *ptr* Pointer to a **PCMarginParams** structure with new document margins.

Return: Nothing.

Interception: Unlikely.

Structures: The **PCMarginParams** structure has the following definition:

```
typedef struct {
      word        PCMP_left;
      word        PCMP_top;
      word        PCMP_right;
      word        PCMP_bottom;
} PCMarginParams;
```

■ MSG_PRINT_CONTROL_GET_DOC_MARGINS

```
void     MSG_PRINT_CONTROL_GET_DOC_MARGINS(
         PCMarginParams *ptr);
```

Use this message to get the present values for the document margins. It returns four integers. These integers represent the left, top, right, and bottom margins, expressed in typographer's points.

Source: Unrestricted.

Destination: Any PrintControl object.

Parameters: *ptr* Pointer to a **PCMarginParams** structure which will hold return value.

Return: Nothing returned explicitly.

ptr Structure filled in with document margins.

Interception: Unlikely.

Structures: The **PCMarginParams** structure has the following definition:

```
typedef struct {
      word        PCMP_left;
      word        PCMP_top;
      word        PCMP_right;
      word        PCMP_bottom;
} PCMarginParams;
```

■ MSG_PRINT_CONTROL_SET_DOC_SIZE_INFO

```
void        MSG_PRINT_CONTROL_SET_DOC_SIZE_INFO(
            PageSizeReport *ptr);
```

Use this message to set all of the information about the document size and orientation.

Source: Unrestricted.

Destination: Any PrintControl object.

Parameters: *ptr* Pointer to a **PageSizeReport** structure.

Return: Nothing.

Interception: Unlikely.

Structures: The **PageSizeReport** structure has the following definition:

```
typedef struct {
      dword             PSR_width;
      dword             PSR_height;
      PageLayout        PSR_layout;
      PCMarginParams    PSR_margins;
} PCMarginParams;
```

■ MSG_PRINT_CONTROL_GET_DOC_SIZE_INFO

```
void        MSG_PRINT_CONTROL_GET_DOC_SIZE_INFO(
            PageSizeReport *ptr);
```

Use this message to set all of the information about the document size and orientation.

Source: Unrestricted.

Destination: Any PrintControl object.

Parameters: *ptr* Pointer to an empty **PageSizeReport** structure which the message handler will fill in.

Return: Nothing.

Interception: Unlikely.

Structures: The **PageSizeReport** structure has the following definition:

```
typedef struct {
      dword             PSR_width;
      dword             PSR_height;
      PageLayout        PSR_layout;
      PCMarginParams    PSR_margins;
} PCMarginParams;
```

17.4.5 Print Output Object

`PCI_output, MSG_PRINT_START_PRINTING, MSG_PRINT_VERIFY_PRINT_REQUEST, MSG_PRINT_NOTIFY_PRINT_DB`

Some object is going to describe the print jobs to the Print Control. When instantiating the Print Control, set the chosen object's name in the *PCI_output* field:

```
PCI_output = <yourObject>;
```

There are three messages the Print Control may send to its output: MSG_PRINT_START_PRINTING, MSG_PRINT_VERIFY_PRINT_REQUEST, and MSG_PRINT_NOTIFY_PRINT_DB. The Print Output must respond to MSG_PRINT_START_PRINTING, which is the signal that it's time to describe a print job. The Print Output also must respond to MSG_PRINT_VERIFY_PRINTING if the PrintControl's PCA_VERIFY_PRINT bit has been set. Otherwise, it need not handle this message. Finally, the Print Output may have a handler for the MSG_PRINT_NOTIFY_DB, a message the PrintControl will send whenever the Print dialog box comes up or goes away.

17.4.5.1 The Print Method

Assuming your geode doesn't try to do anything fancy with its own printing UI or weird scheduling, probably the most complicated thing you'll have to do when adding printing capability to your geode is write a Print Method.

This message must be handled by the Print Output object, though the handler probably won't be too complicated. In its simplest form, a MSG_PRINT_START_PRINTING handler could just call some graphics commands and finish off by sending MSG_PRINT_CONTROL_PRINTING_COMPLETED to the PrintControl.

The message is accompanied by a pointer back to the Print Control and a GString handle. Any graphics commands drawn to the GString handle will be retained and will become part of the print job.

Before looking at examples, be warned that there are some requirements that every Print Method must meet:

- Be sure the document size has been set correctly. Before the job is finished and you send MSG_PRINT_CONTROL_PRINTING_COMPLETED, you must be certain the document's size and margins have been set somewhere. This may have been set using the *PCI_docSizeInfo* field when defining the Print Control. If you haven't already let the Print Control know about the document size elsewhere, then you must send a MSG_SPOOL_PRINT_CONTROL_SET_DOC_SIZE, probably either in the MSG_PRINT_START_PRINTING or wherever your geode sets up document information in general.
- Be certain the number of pages to be printed has been correctly set. If your application hasn't already set the page range (when the Print Control was created or when the Print dialog box goes up are popular times for this), send a MSG_PRINT_CONTROL_SET_TOTAL_PAGE_RANGE to your Print Control. Make sure you do it before sending in the MSG_PRINT_CONTROL_PRINTING_COMPLETED.
- Be sure the document name has been set. If you use a GenDocumentGroup as your *PCI_docNameOutput*, this will be taken care of for you. Otherwise, some time before the MSG_PRINT_CONTROL_PRINTING_COMPLETED is sent, your *PCI_docNameOutput* object is going to be asked for the document name, and the job won't be spooled until your *PCI_docNameOutput* sends the correct name back to the Print Control.
- The Print Method should end each page with a **GrNewPage()**; the last thing drawn before the print job ends is a **GrNewPage()**.
- Finally, when the Print Method is finished describing the print job, it must end with a MSG_PRINT_CONTROL_PRINTING_COMPLETED. Otherwise, the Print Control has no way of knowing the job is ready.

When the Print Control receives the message MSG_PRINT_CONTROL_PRINTING_COMPLETED, it responds by cleaning up and sending the print job to the spooler.

Your handler can send a MSG_PRINT_CONTROL_REPORT_PROGRESS to the Print Control if it is spooling a large job and wants to reassure the user. If something goes wrong, or if MSG_PRINT_CONTROL_REPORT_PROGRESS returns a signal indicating the user wishes to cancel, MSG_PRINT_START_PRINTING should send a MSG_PRINT_CONTROL_PRINTING_CANCELLED instead.

■ MSG_PRINT_START_PRINTING

```
void    MSG_PRINT_START_PRINTING(
        optr                printControlOD,
        GStateHandle        gstate);
```

The handler for this message should call a number of graphics routines, using the passed GState. When done with graphics routines, the handler should send a MSG_PRINT_CONTROL_PRINTING_COMPLETED (or MSG_PRINT_CONTROL_PRINTING_CANCELED) to the Print Control. The handler may have to accomplish other tasks; see the above bulleted list (the one marked as “the most used and useful section of this chapter”).

Source: PrintControl object.

Destination: The object specified in *PCI_output*.

Parameters:

printControlOD	The optr of the PrintControl object.
gstate	The GState handle to draw to.

Return: Nothing.

Interception: The Print output object must intercept this message, build the print job by drawing to the passed GState, then send MSG_PRINT_CONTROL_PRINTING_COMPLETED to the object in *printControlOD*.

17.4.5.2 Verifying User Choices

You may write a handler for MSG_PRINT_VERIFY_PRINT_REQUEST, which will be sent to the Print Output if the PCA_VERIFY_PRINT bit has been set. The message will be sent after the user has dismissed the Print dialog box, after the system has had a chance to make sure the document fits on the paper but before the document is actually spooled.

This message gives the application a chance to check out the state of the UI and make sure that the user’s choices are valid. Note that the PrintControl does its own checking to make sure that the document will fit on the page. After examining the UI, the handler must send back a MSG_PRINT_CONTROL_VERIFY_COMPLETED, passing the argument to say whether it's okay to print.

■ MSG_PRINT_VERIFY_PRINT_REQUEST

```
void    MSG_PRINT_VERIFY_PRINT_REQUEST(
        optr   printControlOD);
```

The handler for this message should make whatever checks are necessary to make sure that the user has made valid choices with the printing UI. The handler should then send back an indication of whether the user's choices are all right by means of a MSG_PRINT_CONTROL_VERIFY_COMPLETED.

Source: PrintControl object.

Destination: The object specified in *PCI_output*.

Parameters: *printControlOD* The optr of the PrintControl object.

Return: Nothing explicitly. However, the handler for this message must send a MSG_PRINT_CONTROL_VERIFY_COMPLETED back to the print control.

Interception: If you've set the PCA_VERIFY_PRINT flag, you must intercept this message correctly or else the print job will never start.

17.4.5.3 Dialog Box Notification

The Print Control will send the Print Output a MSG_PRINT_NOTIFY_DB every time the Print dialog box comes up or goes away. Note that this message allows the application to update the page range and print group's UI at only those times that updates are needed.

This message will arrive with one argument, a pointer back to the PrintControl object that sent it. The handler for this message might wish to update the page range information using MSG_PRINT_CONTROL_SET_TOTAL_PAGE_RANGE and MSG_PRINT_CONTROL_SET_SELECTED_PAGE_RANGE.

■ MSG_PRINT_NOTIFY_PRINT_DB

```
void      MSG_PRINT_NOTIFY_PRINT_DB(
          optr                printControlOD,
          PrintControlStatus  pcs);
```

The handler for this message can do any almost anything; this message signals that the Print dialog box has just come up or gone away.

Source: PrintControl object.

Destination: The object specified in *PCI_output*.

Parameters: *printControlOD* The optr of the PrintControl object.

pcs The status of the print control; either PCS_PRINT_BOX_VISIBLE or PCS_PRINT_BOX_NOT_VISIBLE.

Return: Nothing.

Interception: Some Print Output objects will intercept this message to set page ranges.

17.4.6 Document Name Output

PCI_docNameOutput, MSG_PRINT_GET_DOC_NAME

The *PCI_docNameOutput* field must contain the optr of an object which can tell the Spooler the document's name. The document name is displayed on the Printer Control Panel and is how the user can figure out which job is which on the queue. If a GenDocumentGroup is set as the Print Control's *PCI_docNameOutput*, then that object will automatically do the right thing. If you use some other type of object, it must be prepared to supply document names on demand. This demand will come in the form of a MSG_PRINT_GET_DOC_NAME.

This message comes with a pointer back to the Print Control. The handler should respond by sending a MSG_PRINT_CONTROL_SET_DOC_NAME with a string containing the name.

■ MSG_PRINT_GET_DOC_NAME

```
void      MSG_PRINT_GET_DOC_NAME(
          optr   printControlOD);
```

If your *PCI_docNameOutput* object is not a GenDocumentGroup, then it must have a handler for this message.

Source: PrintControl object.

Destination: The object specified in *PCI_docNameOutput*.

Parameters: *printControlOD* The optr of the PrintControl object.

Return: Nothing returned explicitly, but should send a MSG_PRINT_CONTROL_SET_DOC_NAME back to the Print Control.

Interception: If receiving object is a GenDocument, unlikely. Otherwise, interception is necessary.

17.4.7 The Default Printer

`PCI_defPrinter, MSG_PRINT_CONTROL_SET_DEFAULT_PRINTER, MSG_PRINT_CONTROL_GET_DEFAULT_PRINTER`

Advanced Topic

As this documentation has already stressed, one of the printing system's more important features is that it is device independent. However, certain special-purpose applications could conceivably depend on the user's printer type, and those applications may have use for the Print Control's *PCI_defPrinter* field. When the Print Control first appears, it will normally have the system default printer selected. Using the *PCI_defPrinter* option, another printer might be selected. The user may of course override the default printer and select another.

You can set or retrieve the value in *PCI_defPrinter* with the messages listed above. For information on them, see "Working with Instance Data" on page 1081.

There are utility routines to get information about the system default printer. There are commands to retrieve information about each of the installed printers so that the application may find out whether the desired type of printer is installed and, if so, what its printer number is. These routines are described in section 17.7.2.4 on page 1097.

◆Objects

17.4.8 Adding UI Gadgetry

ATTR_PRINT_CONTROL_APP_UI

Advanced Topic

If the Print Control supplies UI that meets your geode's needs, feel free to skip this section. On the other hand, if you've decided you want a non-standard print trigger or want some specialized gadgetry to appear in the application's Print dialog box, it is possible to create objects and tell the Print Control to work with them. Make sure that any objects you want to use in this fashion aren't part of a generic or visual tree already. Also, these objects must be set not usable (~GS_USABLE).

Normally the user has two ways to print up a Print dialog box: the User Print Trigger or the User Fax Trigger, both appearing in the File menu. You may specify a special trigger to be used in place of the standard trigger provided by the Print Control. Normally the default trigger is not only sufficient but preferable, if only because it will automatically display with the correct appearance for the specific UI. To define your own trigger object, create the object(s), setting the correct instance data. Then, when instantiating your Print Control, include an ATTR_GEN_CONTROL_APP_UI:

```
ATTR_GEN_CONTROL_APP_UI = {yourTrigger};
```

When this option is used at all, normally the trigger object is a simple trigger. The action descriptor of this application-defined trigger should be

```
GTI_output = <your_PrintControl>;
GTI_method = MSG_SPOOL_PRINT_CONTROL_INITIATE_PRINT;
```

By working with the control's features, you may remove either or both of the default triggers.

The Print Control's vardata field ATTR_PRINT_CONTROL_APP_UI allows application-specific gadgetry to appear in the Print dialog box. Create the object(s) to be included and set their instance data, then use the ATTR_PRINT_CONTROL_APP_UI field to signal that the Print Control should include the gadgetry.

```
ATTR_PRINT_CONTROL_APP_UI = {yourPrintGroup};
```

When this option is used at all, usually a GenInteraction serves as the print group object, with children appropriate to the task at hand. This generic tree must not be destroyed unless the vardata field has been removed.

17.5 Print Control Messages

Now you have a rather good idea of what's required to set up a Print Control. Once it's been set up, your geode may send the following messages to it.

17.5.1 Common Response Messages

```
MSG_PRINT_CONTROL_VERIFY_COMPLETED,
MSG_PRINT_CONTROL_SET_DOC_NAME
```

These messages are normally used only to respond to messages sent out by the Print Control.

■ MSG_PRINT_CONTROL_VERIFY_COMPLETED

```
void	MSG_PRINT_CONTROL_VERIFY_COMPLETED(
	Boolean continue);
```

If you have decided to verify the user's Print dialog box choices, you must send this message back to the Print Control in response to MSG_PRINT_VERIFY_PRINT_REQUEST. If *true* is passed, the Print Control will continue preparing the job for printing. If the value passed is *false*, the print will be cancelled. It is up to the application to explain the problem to the user.

Source: Unrestricted, normally the *PCI_output* object.

Destination: Any PrintControl object.

Parameters: *continue* Flag signalling whether printing should continue or be cancelled.

Return: Nothing.

Interception: Unlikely.

■ MSG_PRINT_CONTROL_SET_DOC_NAME

```
void	MSG_PRINT_CONTROL_SET_DOC_NAME(
	char	* string);
```

This message must be sent by the *PCI_docNameOutput* before the print job can be spooled. It is sent in response to a MSG_PRINT_GET_DOC_NAME from the Print Control. The *string* argument should be something that will allow the user to recognize the print job in the Printer Control Panel.

Source: Unrestricted—normally the *PCI_docNameOutput* object.

Destination: Any PrintControl object.

Parameters: *string* The document's name, as a null-terminated string. Maximum length of this string is FILE_LONGNAME_LENGTH+1.

Return: Nothing.

Interception: Unlikely.

17.5.2 Flow of Control Messages

```
MSG_PRINT_CONTROL_PRINTING_COMPLETED,
MSG_PRINT_CONTROL_REPORT_PROGRESS,
MSG_PRINT_CONTROL_PRINTING_CANCELLED,
MSG_PRINT_CONTROL_REPORT_STRING,
MSG_PRINT_CONTROL_INITIATE_PRINT, MSG_PRINT_CONTROL_PRINT
```

Sending these messages to the Print Control signals that a new stage of printing is ready to begin.

■ MSG_PRINT_CONTROL_PRINTING_COMPLETED

```
void	MSG_PRINT_CONTROL_PRINTING_COMPLETED();
```

This message signals the Print Control that the application is finished describing the print job. The application must send this message (or a MSG_PRINT_CONTROL_PRINTING_CANCELLED) some time after receiving a MSG_PRINT_START_PRINTING.

Source: The object that received MGS_PRINT_START_PRINTING.

Destination: Any PrintControl object.

Parameters: None.

Return: Nothing.

Interception: Unlikely, unadvised.

■ MSG_PRINT_CONTROL_REPORT_PROGRESS

```
Boolean  MSG_PRINT_CONTROL_REPORT_PROGRESS(
         PCProgressType progress,
         int    pageOrPercent);
```

This message may be sent between the time the Print Output receives a MSG_PRINT_START_PRINTING and the time it sends a MSG_PRINT_CONTROL_PRINTING_COMPLETED. Its arguments include a progress type (page number or percentage) and a number representing the progress. It returns a Boolean value, which will normally be *true* but will be *false* if the user wishes to cancel printing.

The application should send out this message periodically over the course of a long print job, probably about once per page. Progress may be reported as a page number, a percentage of the job completed, or as a null-terminated string.

If the return value is *false*, the application should stop drawing to the print GString immediately and send the Print Control a MSG_PRINT_CONTROL_PRINTING_CANCELLED. The return value may be safely ignored, but if the user wishes to cancel printing, it's polite to cancel immediately instead of making him wait until the MSG_PRINT_START_PRINTING has finished describing the entire job.

Source: The object that received MGS_PRINT_START_PRINTING when building a time-consuming job.

Destination: The PrintControl object specified by MSG_PRINT_START_PRINTING.

Parameters: *progress* How progress is being reported, by page (PCPT_PAGE) or percent (PCPT_PERCENT).

pageOrPercent How much progress has been made.

Return: Flag signalling that printing should continue. If the flag is *false* (i.e. zero), then the user wants to cancel printing, and it would be polite to stop.

Interception: Unlikely.

Structures: The **PCProgressType** enumeration is defined:

◆Objects

```
typedef enum {
 PCPT_PAGE,
 PCPT_UNUSED1, /* Unused type */
 PCPT_PERCENT,
 PCPT_UNUSED2, /* Unused type */
 PCPT_TEXT
} PCProgressType;
```

■ MSG_PRINT_CONTROL_REPORT_PROGRESS_STRING 17.5

```
@alias(MSG_PRINT_CONTROL_REPORT_PROGRESS) \
Boolean  MSG_PRINT_CONTROL_REPORT_PROGRESS_STRING(
         PCProgressType progress,
         Chars  *progressString);
```

This message may be sent between the time the Print Output receives a MSG_PRINT_START_PRINTING and the time it sends a MSG_PRINT_CONTROL_PRINTING_COMPLETED. Its arguments include a text string giving some indication of progress. This message returns a Boolean value, which will normally be *true* but will be *false* if the user wishes to cancel printing.

The application should send out this message periodically over the course of a long print job, probably about once per page. Progress may be reported as a page number, a percentage of the job completed, or as a null-terminated string.

If the return value is *false*, the application should stop drawing to the print GString immediately and send the Print Control a MSG_PRINT_CONTROL_PRINTING_CANCELLED. The return value may be safely ignored, but if the user wishes to cancel printing, it's polite to cancel immediately instead of making him wait until the MSG_PRINT_START_PRINTING has finished describing the entire job.

Source: The object that received MGS_PRINT_START_PRINTING when building a time-consuming job.

Destination: The PrintControl object specified by MSG_PRINT_START_PRINTING.

Parameters: *progress* This must be PCPT_TEXT. The reason that this parameter has only one possible value is that this message is actually an alias of MSG_PRINT_CONTROL_REPORT_PROGRESS, handled by the same assembly routine.

progressString A string of text, giving an indication of progress.

Return: Flag signalling that printing should continue. If the flag is *false* (i.e. zero), then the user wants to cancel printing, and it would be polite to stop.

Interception: Unlikely.

Structures: The **PCProgressType** enumeration is defined:

```
typedef enum {
 PCPT_PAGE,
 PCPT_UNUSED1, /* Unused type */
 PCPT_PERCENT,
 PCPT_UNUSED2, /* Unused type */
 PCPT_TEXT
} PCProgressType;
```

■ MSG_PRINT_CONTROL_PRINTING_CANCELLED

void `MSG_PRINT_CONTROL_PRINTING_CANCELLED();`

The application may send this message to the PrintControl after receiving a MSG_PRINT_START_PRINTING. Do not send both this message and a MSG_PRINT_CONTROL_PRINTING_COMPLETED for a single print job.

The application may send this message to cancel a document while describing a print job, before it would send the MSG_PRINT_CONTROL_PRINTING_COMPLETED.

Source: Unrestricted, probably the *PCI_output*.

Destination: The PrintControl object specified by MSG_PRINT_START_PRINTING.

Parameters: None.

Return: Nothing.

Interception: Unlikely.

■ MSG_PRINT_CONTROL_INITIATE_PRINT

void `MSG_PRINT_CONTROL_INITIATE_PRINT();`

This message, normally sent by the Print trigger in the File menu, is the signal that the Print Control should display the Print dialog box. Geodes with custom print triggers should make sure that those triggers send this message to the Print Control. Geodes using the Print Control's provided print trigger need not send this message.

Source: Unrestricted—typically the user print trigger (or fax trigger).

Destination: Any PrintControl object.

Parameters: None.

Return: Nothing.

Interception: Unlikely.

■ MSG_PRINT_CONTROL_PRINT

```
void    MSG_PRINT_CONTROL_PRINT();
```

This message, normally sent by the Print trigger in the Print dialog box, is the signal that the user has made his printing choices and is ready for the Print Control to verify those choices and spool the job.

Source: Unrestricted—typically the Print trigger in the Print Dialog Box.

Destination: Any PrintControl object.

Parameters: None.

Return: Nothing.

Interception: Unlikely.

17.5.3 Working with Instance Data

```
MSG_PRINT_CONTROL_SET_OUTPUT,
MSG_PRINT_CONTROL_GET_OUTPUT,
MSG_PRINT_CONTROL_SET_DOC_NAME_OUTPUT,
MSG_PRINT_CONTROL_GET_DOC_NAME_OUTPUT,
MSG_PRINT_CONTROL_SET_DEFAULT_PRINTER,
MSG_PRINT_CONTROL_GET_DEFAULT_PRINTER,
MSG_PRINT_CONTROL_GET_PRINT_MODE,
MSG_PRINT_CONTROL_GET_PAPER_SIZE,
MSG_PRINT_CONTROL_GET_PRINTER_MARGINS,
MSG_PRINT_CONTROL_CALC_DOCUMENT_DIMENSIONS,
MSG_PRINT_CONTROL_VERIFY_DOC_MARGINS,
MSG_PRINT_CONTROL_VERIFY_DOC_SIZE
```

The Print Control handles many messages which allow the geode to retrieve and alter the values of the instance data.

■ MSG_PRINT_CONTROL_SET_OUTPUT

```
void        MSG_PRINT_CONTROL_SET_OUTPUT(
            optr   objectPtr);
```

This message, used very rarely, specifies that a different object should become the Print Output. Pass the optr of the object to use.

Source: Unrestricted.

Destination: Any PrintControl object.

Parameters: *objectPtr* The optr of the new *PCI_output*.

Return: Nothing.

Interception: Unlikely.

■ MSG_PRINT_CONTROL_GET_OUTPUT

```
optr        MSG_PRINT_CONTROL_GET_OUTPUT();
```

This message returns the optr of the Print Output.

Source: Unrestricted.

Destination: Any PrintControl object.

Parameters: None.

Return: The optr of the current *PCI_output* object.

Interception: Unlikely.

■ MSG_PRINT_CONTROL_SET_DOC_NAME_OUTPUT

```
void        MSG_PRINT_CONTROL_SET_DOC_NAME_OUTPUT(
            optr   document);
```

This message, used very rarely, specifies that a different object should become the Document Name Output object. Pass the optr of the object to use.

Source: Unrestricted.

Destination: Any PrintControl object.

Parameters: *objectPtr* The optr of the new *PCI_docNameOutput*.

Return: Nothing.

Interception: Unlikely.

■ MSG_PRINT_CONTROL_GET_DOC_NAME_OUTPUT

```
optr    MSG_PRINT_CONTROL_GET_DOC_NAME_OUTPUT();
```

This message returns the optr of the Document Name Output object.

Source: Unrestricted.

Destination: Any PrintControl object.

Parameters: None.

Return: The optr of the Document Name output object.

Interception: Unlikely.

■ MSG_PRINT_CONTROL_SET_DEFAULT_PRINTER

```
void    MSG_PRINT_CONTROL_SET_DEFAULT_PRINTER(
        int    printerNum);     /* -1 for system's default printer */
```

This message sets the application default printer, to be used the next time the print dialog box appears. Pass the printer number of the printer to use. Passing a value of -1 means that the system default printer should be used.

Source: Unrestricted.

Destination: Any PrintControl object.

Parameters: *printerNum* Number of the new default printer.

Return: Nothing.

Interception: Unlikely.

■ MSG_PRINT_CONTROL_GET_DEFAULT_PRINTER

```
int     MSG_PRINT_CONTROL_GET_DEFAULT_PRINTER();
```

Use this message to retrieve the number of the present application default printer. You may not assume that this printer is the one the user is printing on, since the user may have changed to another printer. A return value of -1 means that the application will use the system default printer.

Source: Unrestricted.

Destination: Any PrintControl object.

Parameters: None.

Return: The number of the default printer.

Interception: Unlikely.

■ MSG_PRINT_CONTROL_GET_PRINT_MODE

```
byte    MSG_PRINT_CONTROL_GET_PRINT_MODE();
```

This message retrieves the current user-selected print mode, if any. Its possible return values include zero, meaning no mode has been selected; PM_GRAPHICS_LOW_RES; PM_GRAPHICS_MED_RES; PM_GRAPHICS_HI_RES; PM_TEXT_DRAFT; and PM_TEXT_NLQ.

Source: Unrestricted.

Destination: Any PrintControl object.

Parameters: None.

Return: A **PrinterOutputModes** signalling the printer's quality.

Interception: Unlikely.

Structures: The **PrinterOutputModes** structure has the following definition:

```
typedef ByteFlags PrinterOutputModes;

#define POM_GRAPHICS_LOW 0x10
#define POM_GRAPHICS_MEDIUM 0x08
#define POM_GRAPHICS_HIGH 0x04
#define POM_TEXT_DRAFT 0x02
#define POM_TEXT_NLQ 0x01
```

Each flag indicates the specified quality is available. Two useful masks which have been set up are PRINT_GRAPHICS and PRINT_TEXT.

■ MSG_PRINT_CONTROL_GET_PAPER_SIZE

```
void    MSG_PRINT_CONTROL_GET_PAPER_SIZE(
        PCMargineParams     *retVal);
```

Use this message to retrieve the Print Control's present paper size.

Source: Unrestricted.

Destination: Any PrintControl object.

Parameters: *retVal* An empty **PCMarginParams** structure which the message handler will fill.

Return: Nothing returned explicitly; the *retVal* structure will be filled.

◆ Objects

Interception: Unlikely.

■ MSG_PRINT_CONTROL_GET_PAPER_SIZE_INFO

```
void    MSG_PRINT_CONTROL_GET_DOC_SIZE_INFO(
        PageSizeReport *ptr);
```

Use this message to set all of the information about the document size and orientation.

Source: Unrestricted.

Destination: Any PrintControl object.

Parameters: *ptr* Pointer to an empty **PageSizeReport** structure which the message handler will fill in.

Return: Nothing.

Interception: Unlikely.

Structures: The **PageSizeReport** structure has the following definition:

```
typedef struct {
      dword              PSR_width;
      dword              PSR_height;
      PageLayout         PSR_layout;
      PCMarginParams     PSR_margins;
} PCMarginParams;
```

■ MSG_PRINT_CONTROL_GET_PRINTER_MARGINS

```
void    MSG_PRINT_CONTROL_GET_PRINTER_MARGINS(
        MarginDimensions    *retVal,
        Boolean             setMargins);
```

This message returns the margins enforced by the requested printer. If the boolean argument is *true*, the document margins will be set to be the same as the printer margins.

Source: Unrestricted.

Destination: Any PrintControl object.

Parameters: *setMargins* Pass TRUE to set the document's margins equal to the printer's margins, FALSE to leave the margins as they are.

retVal A pointer to an empty **MarginDimensions** structure to be filled in by the handler.

Return: The printer's enforced margins.

Structures: The following structure keeps track of all four of a document's margins.

```
typedef struct {
      int   leftMargin; /* measured in points */
      int   topMargin;
      int   rightMargin;
      int   bottomMargin;
} MarginDimensions;
```

Interception: Unlikely.

■ MSG_PRINT_CONTROL_CALC_DOC_DIMENSIONS

```
void    MSG_PRINT_CONTROL_CALC_DOCUMENT_DIMENSIONS(
        PageSizeReport *ptr);
```

This message calculates the maximum printable area allowed by the user-selected paper size and printer-mandated margins. This message automatically resets the document size and margins to hold these values.

Note that this message correctly handles rotated pages.

Source: Unrestricted.

Destination: Any PrintControl object.

Parameters: *retValue* A pointer to a **DocumentSize** structure to hold return value.

Return: Nothing returned explicitly.

retValue The pointer to the filled **DocumentSize** structure.

Structures: The following structure holds a document's size, and the margin information used to place the top left point.

```
typedef struct {
      int   leftMargin; /* measured in points */
      int   topMargin;
      int   width;
      int   height;
} DocumentSize;
```

Interception: Unlikely.

◆ Objects

■ MSG_PRINT_CONTROL_CHECK_IF_DOC_WILL_FIT

```
Boolean MSG_PRINT_CONTROL_CHECK_IF_DOC_WILL_FIT(
        Boolean warning);
```

This message returns *true* if the passed document margins will fit on the selected printer.

Source: Unrestricted.

Destination: Any PrintControl object.

17.6

Parameters: *warning* If set, the PrintControl will deliver a warning to the user.

Return: Indication of whether the printer can work with the current margins (*true* if so, *false* if not).

Interception: Unlikely.

17.6 Page Size Control

```
MSG_PRINT_REPORT_PAGE_SIZE, SpoolSetDocSize(),
MSG_PZC_GET_PAGE_SIZE, MSG_PZC_SET_PAGE_SIZE,
```

Thanks to the Spooler, applications don't need to worry about what page size the user is working with. If a document is bigger than a printer's paper, then the Spooler will automatically tile the job onto as may sheets of paper as necessary. However, some applications will need to make the user decide on a page size. Page layout programs need to know what size page the user is working with. Spreadsheet programs need to make sure that cells aren't split in half by a page break.

Applications which ask the user to select a page size should incorporate a PageSizeControl into their generic tree. This generic control object provides a dialog box containing page setup choices for the user.

This dialog box contains UI allowing the user to specify what sort of paper the document is to be printed to: envelope, labels, or regular paper. The user may then specify a set of dimensions within that type; paper would have choices including letter sized, legal sized, and A4. The user can even set up a set of custom page dimensions.

Most applications which include a PageSizeControl will probably keep track of their own page size. In this case, use **SpoolSetDocSize()** to update the PageSizeControl's UI when changing the page size. When the user changes the paper size in the PageSizeControl, the control's output will receive a MSG_PRINT_REPORT_PAGE_SIZE.

Applications which do not keep track of the page size but want to include a PageSizeControl may do so. However, these applications will probably need to find out the page size at a specific time, not just receive notification every time the user changes the page size. This doesn't really fit the normal controller model, and in fact such applications are asking the PageSizeControl to act not like a controller, but like an ordinary piece of UI gadgetry. Setting the PZCA_ACT_LIKE_GADGET flag in the PageSizeControl's *PZCI_attrs* instance field will make the PageSizeControl act like a gadget.

If the PageSizeControl is to act as a controller, it must appear on the GenApplication's GAGCNLT_SELF_LOAD_OPTIONS GCN list, as does the PrintControl.

No matter how an application interacts with its PageSizeControl, it will work with the **PageSizeReport** structure.

```
typedef struct {
      dword             PSR_width;
      dword             PSR_height;
      PageLayout        PSR_layout;
      PCMarginParams    PSR_margins;
} PageSizeReport;
typedef WordFlags PageLayout;
typedef enum {
      PT_PAPER,
      PT_UNUSED1, /* Unused type */
      PT_ENVELOPE,
      PT_UNUSED2, /* Unused type */
      PT_LABEL
} PageType;
/* Specifying a PageLayout is accomplished by
 * OR-ing together the parts of the layout.
 * Exactly what those parts are depends on the
 * PageType: */
```

```
/* Paper Layouts:
 *    (PT_PAPER | (PO_orient << 3) ) */
typedef ByteEnum        PaperOrientation;
#define PO_PORTRAIT           0x00
#define PO_LANDSCAPE          0x01

typedef WordFlags       PageLayoutPaper;
#define PLP_ORIENTATION       0x0008
#define PLP_TYPE              0x0004 /* PT_PAPER */

/* Envelope Layouts:
 * (PT_ENVELOPE | (EP_path << 5) | (EO_ori << 3)) */
typedef ByteEnum        EnvelopePath;
#define EP_LEFT               0x00
#define EP_CENTER             0x01
#define EP_RIGHT              0x02

typedef ByteEnum        EnvelopeOrientation;
#define EO_PORTAIT_LEFT       0x00
#define EO_PORTAIT_RIGHT      0x01
#define EO_LANDSCAPE_UP       0x02
#define EO_LANDSCAPE_DOWN     0x03

typedef WordFlags       PageLayoutEnvelope;
#define PLE_PATH              0x0040
#define PLE_ORIENTATION       0x0010
#define PLE_TYPE              0x0004 /*PT_ENVELOPE*/

/* Label Layouts:
 *    (PT_LABEL | (rows <<8) | (cols << 3)) */
typedef WordFlags       PageLayoutLabel;
#define PLL_ROWS              0x7e00 /* # rows */
#define PLL_COLUMNS           0x01f8 /* # cols */
#define PLL_TYPE              0x0004 /* PT_LABEL */
```

Code Display 17-3 PageSizeControl Features

```
typedef ByteFlags PageSizeControlFeatures;
/* The following flags may be combined with | and &:
        PSIZECF_MARGINS,        ( Set margin sizes )
        PSIZECF_CUSTOM_SIZE,    ( Custom dimension ranges )
        PSIZECF_LAYOUT,
        PSIZECF_SIZE_LIST,
        PSIZECF_PAGE_TYPE       ( Choice of standard dimensions ) */

#define PSIZEC_DEFAULT_FEATURES (PSIZECF_PAGE_TYPE | PSIZECF_SIZE_LIST | \
                                PSIZECF_LAYOUT | PSIZECF_CUSTOM_SIZE)
```

■ MSG_PRINT_REPORT_PAGE_SIZE

```
void    MSG_PRINT_REPORT_PAGE_SIZE(
        PageSizeReport *psr);
```

The page size control's output object should have a handler for this message. The control will send this message whenever the user has changed the page size. The handler may wish to change the bounds of some appropriate object, store the new bounds somewhere, or take some other action.

Source: PageSizeControl object.

Destination: The *GCI_output* object.

Parameters: *psr* A pointer to a **PageSizeReport** structure containing the page type and size.

Return: Nothing.

Interception: The handler for this message should store the page size information. It may set the bounds of one or more visual objects.

Advanced Topic

Only work with instance data if the PageSizeControl is in Gadget mode.

In addition to the standard set of generic Control instance data, the Page Size control includes some of its own. This data, as you might expect, is largely concerned with paper size and setup. The fields are listed in Code Display 17-4.

PageSizeControls use the **PageLayout** data structure to hold and pass layout information such as page orientation. One part of this structure gives the type of page being described: paper, envelope, or label. The other parts of the structure have different meanings depending on the type of page. For paper, the layout keeps track of the orientation. For envelopes, the structure contains both path and orientation information. For labels, this structure keeps track of how many labels are on the sheet in each direction; these numbers are bounded by MAXIMUM_LABELS_ACROSS and MAXIMUM_LABELS_DOWN.

Code Display 17-4 Page Size Control Instance Data

```
@instance PageSizeControlAttrs   PZCI_attrs = 0;
        /* Possible attributes:
         * PZCA_ACT_LIKE_GADGET
         * PZCA_PAPER_SIZE
         * PZCA_INITIALIZE               */

        /* Current page dimensions and type/layout */
@instance dword         PZCI_width = 0;
@instance dword         PZCI_height = 0;
@instance PageLayout    PZCI_layout = 0;
@instance PCMarginParams PZCI_margins = { 0, 0, 0, 0 };

/* The following vardata field is internal: */
@vardata PageSizeControlMaxDimensions TEMP_PAGE_SIZE_CONTROL_MAX_DIMENSIONS;

/* Its structure is defined:
typedef struct {
        dword PZCMD_width; ( maximum width )
        dword PZCMD_height; ( maximum height )
} PageSizeControlMaxDimensions; */


/* Attribute to allow applications to be made aware of every change made
 * to gadgetry in the PageSizeControl. This is especially useful for applications
 * that want to add UI that will be dependent upon the state of this
 * controller. No effort is made to eliminate the redundant output of data. */

@vardata PageSizeControlChanges ATTR_PAGE_SIZE_CONTROL_UI_CHANGES;

/* This field has structure:
        typedef struct {
         optr PSCC_destination; ( destination for message )
         Message PSCC_message; ( message to be sent )
        } PageSizeControlChanges; */

The message should follow the prototype: */
@prototype void PAGE_SIZE_UI_CHANGES_MSG(PageSizeReport _far *psr )
```

PZCI_width, *PZCI_height*
These integers represent the current page dimensions, measured in points. Note that these values may never go outside the bounds described by the constants MINIMUM_PAGE_WIDTH_VALUE, MINIMUM_PAGE_HEIGHT_VALUE, MAXIMUM_PAGE_WIDTH_VALUE, and MAXIMUM_PAGE_HEIGHT_VALUE.

17.6

PZCI_layout
This field contains a **PageLayout** structure describing the current page layout settings.

PZCI_margins
This field contains the page's margins.

The following messages allow applications to work with a PageSizeControl directly, like a regular UI gadget. There is one message which the PageSizeControl will send to its output, and several that any object may send to the controller.

Note that these messages are only useful if the PageSizeControl is operating in gadget mode (i.e. if its PZCA_ACT_LIKE_GADGET bit is set). Otherwise, the control's default behavior would override that requested by these messages.

The **PageSizeControlAttrs** record has three flags:

PZCA_ACT_LIKE_GADGET
The PageSizeControl object will act like any other generic gadget and will not respond to normal controller notifications.

PZCA_PAPER_SIZE
The PageSizeControl object will display paper sizes instead of document sizes. Setting this would be very rare for applications.

PZCA_INITIALIZE
The PageSizeControl object will be initialized to system default values if PZCA_ACT_LIKE_GADGET is also set.

PZCA_LOAD_SAVE_OPTIONS
Allow the controller to load and save the user's options.

Objects

■ MSG_PZC_GET_PAGE_SIZE

```
void    MSG_PZC_GET_PAGE_SIZE(
        PageSizeReport      *psr);
```

This message returns the present page size in terms of width, height, and layout, stored in a **PageSizeReport**. Note that the PageSizeControl will be sending out a MSG_PRINT_REPORT_PAGE_SIZE every time the user applies changes to these values.

Only send this message if the PageSizeControl is operating in gadget mode.

Source: Unrestricted, as long as the PageSizeControl is in gadget mode.

Destination: PageSizeControl object.

Parameters: *psr* A pointer to a **PageSizeReport** structure in which the page size will be returned.

Return: The **PageSizeReport** structure pointed to by *psr* will be filled with the page information.

Interception: Unlikely.

■ MSG_PZC_SET_PAGE_SIZE

```
void    MSG_PZC_SET_PAGE_SIZE(
        PageSizeReport *psr);
```

This message changes the current page size. It takes a **PageSizeReport** data structure containing the new dimensions and layout. The Apply trigger of a Page Size Control normally sends this message, passing the present user page size as arguments.

Only send this message if the PageSizeControl is operating in gadget mode.

Source: Unrestricted, as long as the PageSizeControl is in gadget mode.

Destination: PageSizeControl object.

Parameters: *psr* A pointer to a **PageSizeReport** structure containing page information.

Return: Nothing.

Interception: Unlikely.

17.7 Other Printing Components

Most geodes can fulfill all their drawing needs by working with their Print Control. The system has two other main parts that work with printing: the first is the spool library, which controls scheduling; the second is a collection of printer drivers, which handle interactions between the printing devices and the rest of the system.

17.7

17.7.1 Spooler and Scheduling

`SpoolInfo(), SpoolHurryJob(), SpoolDelayJob(), SpoolModifyPriority(), SpoolVerifyPrinterPort(), SpoolCreateSpoolFile(), SpoolDelJob(), MSG_PRINT_NOTIFY_PRINT_JOB_CREATED`

● Advanced Topic

Geodes may invoke certain routines that affect the Spooler, but for the most part they should avoid this practice. These routines allow the geode to take advantage of some of the Spooler's scheduling capabilities, so the geode can hurry some jobs and delay others. Most applications should not be concerned with the scheduling of printer jobs. If the user thinks that some job should be given a higher priority, he can work with the Printer Control Panel. Many of the following routines require the print job's ID. Applications that will use these routines should have a handler for MSG_PRINT_NOTIFY_PRINT_JOB_CREATED, which has reference material at the end of this section.

For those applications that will be scheduling, the **SpoolHurryJob()** and **SpoolDelayJob()** routines may come in handy. Passed a job ID, these routines move the associated job to the front or back of the queue (see Figure 17-9). **SpoolModifyPriority()** changes the priority of a spool thread so that it may print in the foreground or be moved further into the background. Each function returns a code describing the Spool's Operating Status, alerting the geode if the job wasn't found, the queue was empty, the port couldn't be verified, or even if the operation was successful.

The **SpoolInfo()** routine returns the list of jobs on a queue or information about any individual job. The other kind of information application writers might be interested in is the printer port status. It's sometimes unwise to

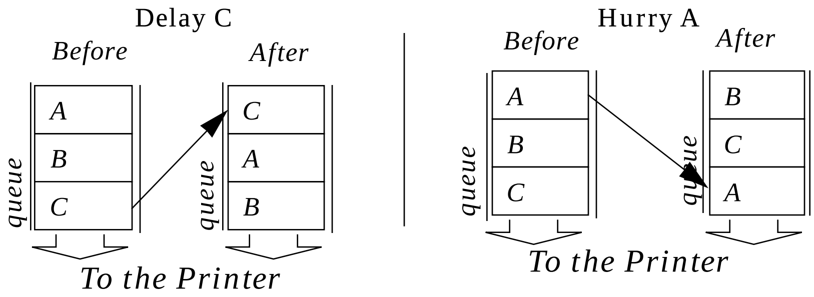


Figure 17-9 *Hurrying and Delaying Jobs*

Print jobs that have already been sent to the printer are not shown here.

spool a job when the printer is having problems, and **SpoolVerifyPrinterPort()** can give some warning about problems.

While the above routines are used only rarely in applications, there are others that are used even less often. **SpoolCreateSpoolFile()** creates a spool file, which may be used to create a pre-generated job for the printer kept handy. Such a file would only be useful to applications that can only print one thing and don't mind leaving around an extra spool file. **SpoolDelJob()** deletes jobs from the queue.

■ MSG_PRINT_NOTIFY_PRINT_JOB_CREATED

```
void    MSG_PRINT_NOTIFY_PRINT_JOB_CREATED(
        word jobID);
```

The PrintControl sends this message when it has sent the job to the print queue. Applications may intercept this message to retrieve the new job's ID number or for any other reason.

Source: PrintControl object.

Destination: *GCI_output* object.

Parameters: *jobID* Print job's ID.

Return: Nothing.

Interception: Applications that want to know the ID of created jobs should intercept this message. Note that this behavior is not encouraged, and most applications will want to leave meddling with a print job's status up to the user's discretion.

17.7.2 Printer Drivers

Printer Drivers are very important but at the same time not something that any geode you write is going to interact with. Printer Drivers take care of translating the device-independent graphics commands into forms that individual models of printers can work with. Drivers also provide the Spooler with information about the printer and provide printer-specific UI to the Print Control.

As long as a geode uses only device-independent graphics commands, that geode need not worry about printer drivers. In fact, the only geodes that require knowledge of printer drivers are those that use Raw Mode (see below) printing and access the printer drivers directly. To incorporate Raw Mode printing into a geode (normally a bad idea), keep reading. Whether or not the geode you're writing will interact with Printer Drivers, you might want to continue reading this section to find out what printer drivers do.

17.7.2.1 Dumb Printers

When dealing with a printer that has no page description language, the Spooler and Printer Driver have to be clever. First the Spooler allocates an offscreen bitmap and plays back the contents of the spool file into the bitmap. It does this through the **vidmem** driver, which has the same interface as a video driver but draws to offscreen bitmaps instead of to a device. Once the bitmap is complete, the Spooler starts feeding the bitmap to the Printer Driver, which translates the bitmap into a series of commands to the printer. If (as is often the case) the printer doesn't have enough memory to absorb a whole page's worth of data at a time, the Printer Driver feeds in one horizontal band, or "swath" of the bitmap at a time. This also allows for better backgrounding: The driver can feed in a swath, then block and allow other threads to complete other tasks until the printer is ready for the next swath.

17.7.2.2 Smart Printers

When dealing with an "intelligent" printer, a printer that has its own page description language, the Spooler bypasses most of the work it has to do when dealing with dumb printers. Since an intelligent printer presumably has some commands with GEOS equivalents, the Spooler doesn't bother with creating a bitmap but just passes the GString on to the Printer Driver. The Printer Driver then builds a page description in the printer's native language based on the contents of the GString.

17.7.2.3 Raw Mode

● Advanced Topic

If you want to use some printer-specific command that doesn't have an equivalent in the GEOS graphics system, you will use raw mode. Raw Mode printing expects that you know what sorts of commands the printer will be expecting. If you try to send PostScript commands to a printer that doesn't understand PostScript, probably the results won't be the desired output.

17.7.2.4 Printer Information and Manipulation

```
SpoolGetNumPrinters(), SpoolGetPrinterString(),
SpoolGetPrinterInfo(), SpoolCreatePrinter(),
SpoolDeletePrinter(), SpoolGetDefaultPrinter(),
SpoolSetDefaultPrinter()
```

● Advanced Topic

If you want to find out something about a printer without accessing the printer driver directly, the spool library provides a number of utility routines that work with printer drivers.

Most of these functions specify which of a user's printers they wish to work with by passing its printer number. The first installed printer will be printer number zero, the second printer will be number one, and so on. Use the **SpoolGetNumPrinters()** routine to find out how many printers have been installed. Once you have this number, you have an upper bound for printer numbers to test. There will always be fewer than MAXIMUM_NUMBER_OF_PRINTERS printers.

The **SpoolGetPrinterString()** and **SpoolGetPrinterInfo()** routines return information about the printer associated with the passed printer number. **SpoolGetPrinterString()** returns the a string containing the name of the printer, such as "HP DeskJet on COM1." The printer string will be of length at most MAXIMUM_PRINTER_NAME_LENGTH. **SpoolGetPrinterInfo()** returns various pieces of information helpful to geodes that work with printers directly.

The **SpoolCreatePrinter()** and **SpoolDeletePrinter()** routines can install and uninstall printers, tasks probably best left to the Preferences Manager. The **SpoolDeletePrinter()** routine uninstalls the printer corresponding to the passed printer number. It is easy to use but not something that most geodes should be doing. **SpoolCreatePrinter()** installs a new printer, returning the new printer's printer number. Note that any geode calling this latter function is going to be expected to provide considerable information about the printer being installed. Again, this is not a routine that many geodes should be using.

SpoolGetDefaultPrinter() returns the number of the system-default printer. **SpoolSetDefaultPrinter()** makes the printer associated with the passed printer number the new system-default printer. Note that these routines are concerned with the system-default printer as opposed to the application-default printer. Most applications won't specify an application specific default printer. For these applications, the system-default printer will be the default. Of course, the user will still be allowed to select any printer, ignoring the default.

17.7.3 Page Size Related Routines

`SpoolGetNumPaperSizes(), SpoolGetPaperString(), SpoolConvertPaperSize(), SpoolCreatePaperSize(), SpoolDeletePaperSize(), SpoolGetPaperSizeOrder(), SpoolSetPaperSizeOrder()`

Advanced Topic

Most geodes won't be too concerned with what sort of page size choice the user has made. If the document fits on the page, all's well. If the document doesn't fit, the Spooler will tile the job using only as many pieces of paper as necessary. If the document size should be the same as the page size, the geode can send a MSG_PRINT_CONTROL_CALC_DOCUMENT_DIMENSIONS to the print control.

There are several routines which deal with page sizes. Most applications have no reason to call these functions, and we will not discuss them in detail here. Some general notes are in order for those programmers who might work with these functions, though.

It takes two indexes to access a page size. The first is the **PageType**: paper, envelope, or label. Within each type, there are up to MAX_PAPER_SIZES page sizes. To find out the number of page sizes presently defined for a given type, use the **SpoolGetNumPaperSizes()** routine. To find the number of a paper size associated with a **PageType** and a given set of dimensions, use the **SpoolConvertPaperSize()** routine. For the text string describing the size of the page, call **SpoolGetPaperString()** (the buffer to hold the string should be of size MAX_PAPER_STRING_LENGTH.

To find out the order in which the page sizes of a given type are stored, call **SpoolGetPaperSizeOrder()**. You may give a new order by calling **SpoolSetPaperSizeOrder()**. This will affect the order in which page sizes are presented in a page size control.

To create a new page size, call **SpoolCreatePaperSize()**. This routine takes a page type, a set of dimensions, and a string to describe those dimensions to the user. It returns a new page size number. To destroy a page size, call **SpoolDeletePaperSize()**.

17.8 Debugging Tips

Though GEOS provides an easy-to-use printer interface, mistakes are sure to happen. You may find the following information useful in catching your mistakes or for testing rigorously.

Under Swat, you can see the various threads the Spooler works with. There is always one spool thread running in the background. When the first job is queued on a printer, you can see that a new thread is created for that printer. When the last job for a printer is completed, the thread exits again.

Here are the most common printing mistakes:

- Forgetting to add the Print Control object to the application's Active List (the MGCNLT_ACTIVE_LIST GCN list).
- Not setting the document name (see section 17.4.6 on page 1073).
- Setting the number of pages (page range) incorrectly.

17.8

Objects

Graphic Object Library

18

The grobj (short for "Graphic Object," sometimes capitalized GrObj) library allows an application to include a graphic layer. This graphic layer provides an interactive graphics environment, allowing the user to create, move, and alter shape objects as in GeoDraw. This graphic layer might allow the user to add graphic elements to a document. The application could even use a modified graphic layer to represent the document, perhaps subclassing one of the "shape" object classes to provide application-specific features.

The Graphic Object library works well with the ruler library. If your geode includes both a graphics layer and one or more rulers, it is possible to alert the graphic layer to the presence of the ruler. The graphic layer will automatically work with rulers, relaying mouse events and even respecting ruler-imposed mouse constraints.

This chapter was written expecting that the reader would look over the DupGrObj sample application. This application shows a sparse graphical layer setup and includes the most basic necessary objects.

18.1 Setting Up the Objects

Several objects work together to manage a graphics layer. Each shape is represented by an object, and there are some objects which act together to manage these shapes. These manager objects are probably the only objects your geode will interact with directly.

The objects which manage the graphic layer are the GrObjBody, GrObjHead, and GrObjAttributeManager objects. Any application which uses one or more graphics layers must include a GrObjHead object. This object keeps track of data which is shared by all graphics layers in an application. The most important piece of information is the current tool. Users might be disconcerted if the drawing tool suddenly changed whenever the mouse moved between graphical documents, so the GrObjHead makes sure that only one tool is in use at a time.

Each "graphic layer" is actually a GrObjBody object. The GrObjBody manages the layer's "shape objects," managing their memory and storing them in a drawing-order list. The GrObjBody acts as a sort of communication nexus. When a controller wants to change a shape's line color, the message is actually sent to the GrObjBody, which in turn figures out which object should change and forwards the message accordingly. The GrObjBody can relay classed events to its associated GrObjHead, rulers, or attribute manager.

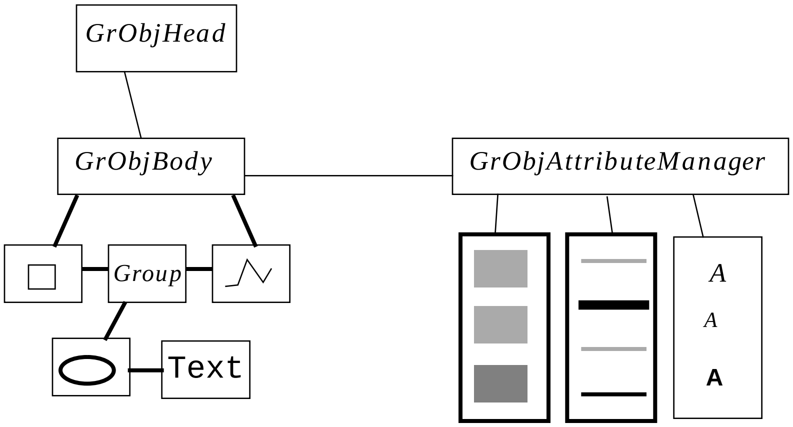


Figure 18-1 *Relations Between Objects*

If your application is including a graphic layer in each document, then there should probably be one GrObjBody for each document. The GrObjBody object must be stored in a VM file. **GrObjBodyClass** is a subclass of **VisCompClass**, and may be incorporated as the child of a GenDocument's visual component as with any other visual object. Of course, if the GrObj is part of a GenDocument, it may (and probably should) be stored in the VM file associated with the document.

Quite often, several shapes in a graphics layer will have the same properties. It would be a waste of space if every shape object were to store its own properties—all the shapes with standard thin black borders would be storing redundant information. The properties of a layer's objects are stored in a GrObjAttributeManager (often written "GOAM"), where they may be referenced by tokens. Instead of storing a structure full of property information, each shape keeps track of one or more tokens: a rectangle would have two tokens, one representing its area properties, and one representing its line properties. This mechanism also makes it possible to implement graphical "style sheets." Since the drawing properties for all shapes are stored in one place, it is possible to change the way that several shapes draw by changing data in one place.

Note that if a graphic layer will be saved, its associated attribute manager must be saved as well; without the drawing properties stored in the GOAM, half the information associated with a graphic layer would be lost. The GOAM must be stored in a VM file, the same VM file which contains all managed GrObjBodies.

An attribute manager can manage the attributes of more than one GrObjBody. If the GOAM will be saved, it will store the information associated with all managed graphic layers.

Your application will probably never directly interact with the "shape objects" maintained by the GrObjBody. These are objects of the **GrObjClass**, often referred to as "GrObjs."

GrObjs are also used to represent groups of fused objects, by means of the **GroupClass** subclass. Thus, a group may be manipulated as a single object, with moving and stretching handles like any shape object.

Spline, bitmap, and text GrObjs are implemented using visual objects in conjunction with some special graphic objects. Each object of this sort is actually made up of two objects, known as a guardian and a ward. The guardian object is a graphic object which has been subclassed to work with visual objects. Each guardian object keeps track of a ward visual object. The guardian object intercepts GrObj-related messages, translates them into the appropriate visual object messages and relays them to its ward.

The grobj library defines several controller classes which your application may use to allow the user to carry out certain standard graphic object operations including changing the drawing tool, changing area properties, and nudging graphic objects.

Applications using the grobj library are also encouraged to use the ruler library. The ruler library adds rulers, grids, and guidelines to a graphic layer. Note that the GrObjBody correctly directs classed events to an associated ruler, so there is no need to use the classed event redirection code normally used with VisRulers (as long as the GrObjBody is the target of the classed event).

18.1.1 Initializing the Objects

Most of the graphic objects you will be including will be able to set up their own instance data. However, your application must send certain messages to hook up these objects to their UI. These messages must be sent as soon as the appropriate objects are created, whether they are declared or created dynamically.

The GrObjAttributeManager needs notification that it should initialize the data structures in which it will store the attributes. As soon as the GrObjBody is created, something should send a MSG_GOAM_CREATE_ALL_ARRAYS to the Attribute Manager, where that something is probably the same object responsible for creating the GrObjBody. In the DupGrObj sample application, this is done in the handler for MSG_GEN_DOCUMENT_INITIALIZE_DOCUMENT_FILE.

If the GrObjBody and GrObjHead are to communicate properly, When attaching the GrObjBody to the UI, you must send it a MSG_GB_ATTACH_UI, passing the optr of the GrObjHead. The GrObjBody and GrObjHead will then work together to set up the necessary communication links.

18.1.2 GrObj in a GenDocument

The GrObj library has several optimizations for those geodes which will be using graphic layers as documents, and most applications will probably use graphic objects in this context. The DupGrObj sample application is a good example of how to set up the main objects for a typical GenDocument setup. When setting up a GenDocument, you must consider which objects should appear in the document's UI, and which of those objects must have their state saved when the document is saved. Note that if you are going to include a graphics layer in a GenDocument, you should be familiar with the GenDocument class. The rest of this section won't make much sense to you otherwise.

The GrObjHead object probably shouldn't be controlled by the GenDocument in any way. This object should not be part of any generic or visual tree (not surprising, as it is neither a generic nor visual object). Its state does not need to be saved with the document; the current tool isn't really part of the document.

Similarly, the various GrObj controllers should not be saved with the document nor should they be in the document's UI. Normally, these UI components are children of the primary, because their influence goes beyond individual documents. Their state does not need to be saved with the document, since values are updated (and thus any saved state values wiped out) whenever a new shape object is selected.

The GrObjBody and the GrObjAttributeManager must be saved with the document, because they manage the graphic layer. The GrObjBody should be incorporated into the GenDocument's visual tree in the usual way.

If your environment includes VisRulers, you may wish to include these in the document's VM file. Remember that the rulers are in charge of storing grid and guideline information, and if their state is not saved, this information will be lost. However, the RulerView and RulerContent used to contain each VisRuler object contain no useful state information, and thus do not need to be saved.

DupGrObj also illustrates the proper time to send GrObj initialization messages under the GenDocument model. Notice the MSG_GEN_DOCUMENT_INITIALIZE_DOCUMENT_FILE handler sends a MSG_GOAM_CREATE_ALL_ARRAYS immediately after attaching the resource to a VM block. The MSG_GEN_DOCUMENT_ATTACH_UI_TO_DOCUMENT sends a MSG_GB_ATTACH_UI.

18.2 Managing a Graphic Layer

There is no single best way to manage a graphic layer. Different applications work with the GrObj library in different ways. Some applications may use documents that consist only of a graphic layer. For these applications, the main concerns will probably include customizing the behavior of the GrObjBody and one or more shape classes. Other applications may wish to use an unmodified graphic layer in conjunction with some other object which could represent the non-graphical component of the data. Applications following this approach will have varying concerns depending on the relation between the graphic layer and other objects.

Many applications follow a model by which the GrObjBody acts as a graphic layer above some other object, such as a text object or spreadsheet object. The grobj library is well suited to this task, but some problems arise to which the solutions will vary from application to application. Some of the more common issues are discussed below, along with possible solutions.

18.2.1 Selection

If you're maintaining a graphic layer above some other object, then you may want the user to be able to click in the document area and have the application automatically know where to send the event. Different applications will probably want to handle this in different ways. What happens when the user does a click drag that starts out in empty space? The GrObj would select everything inside the drag rectangle (or perhaps create a new shape object). But the layer underneath might want to do something (e.g. a VisText might want to select a range of text). Different applications would handle this message in different ways.

One typical approach is to add a new tool item to the graphic tool palette for working with data in the non-graphic layer. If the geode includes a GrObjToolControl, then set up a GrObjToolItem for this tool. There probably won't be a GrObj class associated with this tool item. However, if the user selects this tool, the GrObjHead is still alerted. If an object wants to know if it should grab the focus, it may query the GrObjHead to find out which tool is selected.

When the user has this tool selected, GeoWrite knows that the user is working with the text layer. When one of the normal GrObj tools is selected, then the user is working with graphics.

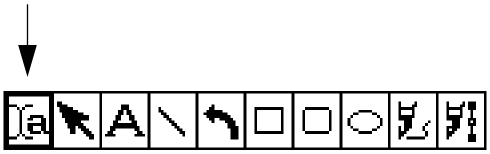


Figure 18-2 *Selection with an Extra Tool*

In the case where there is no GrObj class associated with an item in the GrObjToolControl, if the user selects this tool, then the GrObjHead will store a NULL for the present GrObj class. Each time the tool changes (every time the GrObjHead receives a MSG_GH_SET_CURRENT_TOOL), the application may check for a NULL tool class and set the target appropriately.

18.2.2 Creating GrObjs

The user can create shape objects by interacting with the graphics layer using the provided shape classes (e.g., rectangles, splines). The application doesn't need to do any runtime work; this functionality is built in to the GrObjBody. However, many applications will have cause to create objects without the user's help. The chart library does this, creating several objects and including them in the graphic layer.

1 Instantiate the graphic object with MSG_GB_INSTANTIATE_GROBJ.
To create a new graphic object, send the graphic body a MSG_GB_INSTANTIATE_GROBJ. This creates the GrObj in a block managed by the body.

2 Initialize instance data with MSG_GO_INITIALIZE.
You must set the object's initial size and position by sending it a MSG_GO_INITIALIZE. The object will respond to this message by setting its size and position accordingly. Also, the object will set up its attributes to the default values. At this time, guardian objects will create their visual wards.

3 Initialize other data as desired.
If you want to change anything about the object before it appears, this would be the time to do it. Send messages to the graphic object to change its colors, locks, or any other attribute. If you wish to send messages to the visual object within a text, spline, or bitmap object, send a MSG_GOVG_GET_VIS_WARD to get the OD of the visual object, and then send messages to it.

4 Send the GrObj a MSG_GO_NOTIFY_GROBJ_VALID.
Until the GrObj knows it's valid, it won't be drawn or detectable. When you've finished setting up the object's data, let it know it's ready with a MSG_GO_NOTIFY_GROBJ_VALID.

5 Add the GrObj to the GrObjBody with MSG_GB_ADD_GROBJ.
The final step is to attach the GrObj to the body. After this message is sent, the object will be drawn and the user will be able to interact with it.

To force a redraw after adding the object call MSG_GB_ADD_GROBJ_THEN_DRAW instead of MSG_GB_ADD_GROBJ.

18.2.3 Action Notification

Some applications might want to know when the user is modifying a shape object. You may specify an "action notification" for each graphic object, so that a specified object will receive a specified message whenever that object is modified. The message number and object's optr are stored in the GrObj's ATTR_GO_ACTION_NOTIFICATION vardata field. Note that the object must be stored in the GrObjBody's VM file or in a resource so that the vardata field may be relocated. The action notification may either be declared in a GrObj's ATTR_GO_ACTION_NOTIFICATION field, or may be set dynamically with MSG_GO_SET_ACTION_NOTIFICATION_OUTPUT.

Thus, if an application were using a graphics layer to model the inner workings of a complicated musical instrument, some shape objects might send notification messages to other objects which modeled the instrument's state, and might in turn interact with the sound library. Thus, the user could perhaps produce a sound by manipulating a polygon depicting a spit valve.

If for some reason an application always needed to know the combined surface area of all GrObjs in a graphic layer, then all GrObjs might have the process as their output descriptor, and the process could handle the action notification message by checking for resize actions and adjusting the area total accordingly.

Actually, the case in which all GrObjs will want to send action notification to a common output descriptor comes up often, and there is an optimization. It is possible to set a default action notification output in the GrObjBody. Any objects which don't have their own action notification structure will send notification based on the body's default notification structure. Of course, if neither GrObj nor GrObjBody has a notification structure set up, then no notification will go out. The GrObjBody's default action notification descriptor, if any, is stored in the ATTR_GB_ACTION_NOTIFICATION vardata field. It may be declared in this field, or may be set dynamically by means of MSG_GB_SET_ACTION_NOTIFICATION_OUTPUT. If the GrObjBody will have such an action notification object descriptor, it should have its *GBI_flags* GBF_HAS_ACTION_NOTIFICATION bit set.

Objects

It is possible to suspend an object's notification. By sending MSG_GO_SUSPEND_ACTION_NOTIFICATION (MSG_GB_SUSPEND_NOTIFICATION to suspend notification for all GrObjs managed by a graphic body), a geode may ask that the object not send the notification message until the corresponding MSG_GO_UNSUSPEND_ACTION_NOTIFICATION (MSG_GB_UNSUSPEND_ACTION_NOTIFICATION to affect all objects in a body) is sent. It is a good idea to suspend action notification when handling the notification message; otherwise, any modifications to the GrObj made in response to the notification message would generate another notification message, leading to an infinite loop. The default handlers for these messages account for the case when there is no action notification structure set up.

Notification suspensions may be nested (about 250 times), and the object maintains a count keeping track of how many suspends are in effect. This suspend count is zeroed when the file containing the GrObj is closed. The standard MSG_GO_SET_ACTION_NOTIFICATION_OUTPUT handler also zeroes the suspend count.

You need to declare the notification message and write a handler for it. The message may receive the modified GrObj's OD and a value specifying what sort of action has been performed. The action type is specified by a **GrObjActionNotificationType** value.

```
@message void MSG_MY_NOTIFY (
      optr grobj,
      GrObjActionNotificationType action);
```

Though this notification message does not include a return value, because it passes the GrObj's OD, the notification message's handler may send messages back to the GrObj. If you do this, remember that these messages may themselves constitute "actions" which will generate notifications, and you may wish to suspend notification before sending them.

18.2.4 Locks and Forbidding Actions

To prevent the user from performing certain actions upon a graphic object, set the object's locks accordingly.

- Move
  You may set a lock so that an object may not move. This might come in handy if you wish to include some sort of page header which the user should not be able to move directly.
- Resize
  This lock keeps the user from resizing an object. The chart object uses this lock to keep users from changing charts without working through the chart's data.
- Rotate
  This flag keeps the object from rotating. Some users are confused by rotated text objects, so an application trying to protect users from confusion might set this lock on text objects.
- Skew
  Set this lock to prevent skewing. This is another lock often used with text objects.
- Edit
  Set this flag to prevent the user from editing the innards of a shape object which is based on a visual object. This prevents changing a text object's text, a spline's points, and a bitmap's picture.
- Delete
  This lock keeps the user from deleting an object. This lock is useful for adding page headers to drawings which the user should not be able to remove directly.
- Select
  This lock prevents the user from selecting the object.
- Attribute
  Set this lock to keep the user from changing the object's drawing attributes, such as color or pattern.
- Group
  Set this lock to prevent the user from incorporating the object into a group. Applications which use their own grouping strategy to keep related objects together may wish to set this flag at appropriate times.
- Ungroup
  Set this lock on a group object to prevent the user from breaking it up. Applications which model a complicated object by grouping two or more

simpler objects might want to set this lock to prevent the user from separating the unit.

- Draw
  Set this lock to make the object invisible. Applications which want to hide objects can set this flag at appropriate times.
- Print
  Setting this lock renders the object invisible while printing. Applications which include objects meant as guides probably would set this lock for those objects.

18.2.5 Wrapping

Each graphic layer provides support for applications that will support the idea of “wrapping” data around graphics. However, the exact meaning of wrapping may change depending on other components in the application. Users have a pretty good idea of what is meant by wrapping text around graphic objects (see Figure 18-3). However, other sorts of applications will have other models of how their data should wrap around graphics, if any notion of wrapping applies at all.

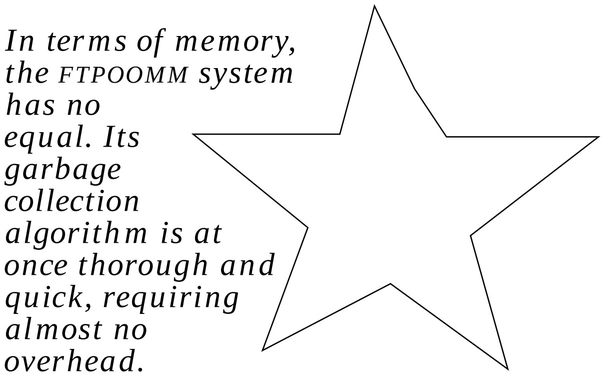


Figure 18-3 *Wrapping Text*

For those applications that will support some form of wrapping, GrObj provides a way for the application to find out what to wrap around. Each GrObj has an attribute in its *GOI_attrFlags* field called GOAF_WRAP. If this bit is set, it means that the GrObj is “solid” and that data should wrap around it. If this bit is not set, then feel free to draw data over the object.

It is possible to draw a graphic layer in such a way that only those GrObjs with the GOAF_WRAP bit set will draw. To draw this way, a geode can send a MSG_GB_DRAW, passing the GODF_WRAP_ONLY bit set. Also, a graphic layer will restrict drawing to wrapping objects by default if its GODF_DRAW_WRAP_TEXT_AROUND_ONLY bit is set.

To determine the bounding path of the region to be wrapped around, draw the graphic layer (with a wrap only option) to a path:

```
GrBeginPath (MyGState, PCT_REPLACE);
@call MyGrObjBody::MSG_GB_DRAW(MyGState,
             0, GODF_DRAW_WRAP_TEXT_AROUND_ONLY);
/* if the GrObjBody’s GBI_drawFlags includes
 * GODF_DRAW_WRAP_TEXT_AROUND_ONLY, we could use
 * either the above or:
 * @call MyGrObjBody::MSG_VIS_DRAW(0, MyGState); */
GrEndPath (MyGState);
```

What the application does with this path is up to you.

18.2.6 Cut, Paste, and Transfer Items

Depending on the relation between graphics and other data in your application, you may end up creating an application-specific clipboard format. The GrObjBody can handle MSG_META_CLIPBOARD_CUT, MSG_META_CLIPBOARD_COPY, and MSG_META_CLIPBOARD_PASTE already. This is fine if the clipboard will never have to store graphical data and some other kind of data at the same time. For instance, if your spreadsheet application wants to link a chart created in a graphic layer to the spreadsheet data which is being charted, then you must create special cut and paste handlers to combine the spreadsheet and graphical information into a single item that may be copied and pasted.

Before attempting such a thing, you should be familiar with clipboard actions, described in “The Clipboard,” Chapter 7 of the Concepts Book. By interacting with the GrObjBody, it is possible to get a VM block transfer item which may be chained together with other VM blocks (representing spreadsheet data or what have you) to form a larger transfer item. Your MSG_META_CLIPBOARD_CUT and MSG_META_CLIPBOARD_COPY handlers can send a MSG_GB_CREATE_TRANSFER to the GrObjBody, creating a transfer item incorporating the data of all selected GrObjs. This transfer item can then be combined with other data before being passed on to the clipboard.

The MSG_META_CLIPBOARD_PASTE handler can extract the GrObj transfer item from the combined data, then pass this data on to the target GrObjBody as the parameter of a MSG_GB_REPLACE_WITH_TRANSFER.

If you are creating a GrObj subclass, then by default the graphic object will include its instance data whenever it is cut or pasted. If objects of this class need to include information other than the instance data, then subclass the behavior of MSG_GO_CREATE_TRANSFER to construct the VM chain to hold all relevant data. Also, subclass MSG_GO_REPLACE_WITH_TRANSFER to correctly extract and apply the transfer item information.

18.3 GrObj Controllers

The grobj library provides a full suite of controllers which allow the user to access some of the graphic library functionality which cannot be accessed by merely clicking within the GrObjBody. Most applications which work with a graphic layer should include at least the GrObjToolControl, and most of the following controllers will be useful to all applications that work with the GrObj.

Most of these controllers will send their messages to the target. As long as the GrObjBody is at the target, it will then relay these messages on to those GrObjs which are selected.

18.3.1 GrObjToolControl

The GrObjToolControl allows the user to change tools. If you wish your graphic layer to disallow certain tools, then remove the appropriate features.

To allow an extra tool, you may take advantage of a special piece of vardata set up for this control.

The GrObjToolControl is something of an anomaly among GrObj-related controllers. Instead of interacting with the targeted GrObjBody, probably it should instead send all messages to the GrObjHead.

18.3

Code Display 18-1 GrObjToolControl Features

```
/* GrObjToolControlClass is a subclass of GenControlClass.
 * Add your GrobjToolControl to GAGCNLT_SELF_LOAD_OPTIONS. */

typedef WordFlags GOTCFeatures;
/* These flags may be combined with | and &:
        GOTCF_PTR,
        GOTCF_ROTATE_PTR,
        GOTCF_ZOOM,
        GOTCF_TEXT,
        GOTCF_LINE,
        GOTCF_POLYLINE,
        GOTCF_POLYCURVE,
        GOTCF_SPLINE,
        GOTCF_ARC,
        GOTCF_RECT,
        GOTCF_FOUNDED_RECT,
        GOTCF_ELLIPSE */

#define GOTC_DEFAULT_FEATURES (GOTCF_PTR | GOTCF_TEXT | GOTCF_LINE | GOTCF_ARC | \
                GOTCF_POLYLINE | GOTCF_ROTATE_PTR | GOTCF_RECT | GOTCF_ZOOM | \
                GOTCF_ROUNDED_RECT | GOTCF_ELLIPSE | GOTCF_POLYCURVE |GOTCF_SPLINE)
#define GOTC_DEFAULT_TOOLBOX_FEATURES (GOTC_DEFAULT_FEATURES)

/*
        GrObjToolControlClass also includes a piece of vardata used to help
                applications that will include one or more extra tools. */
        @vardata word ATTR_GROBJ_TOOL_CONTROL_POSITION_FOR_ADDED_TOOLS

/*      The UI for this tool should be provided in an
        ATTR_GEN_CONTROL_APP_TOOLBOX_UI vardata field.
*/
```

Objects

18.3.2 GrObjStyleSheetControl

The GrObjStyleSheetControl allows the user to create graphical “style sheets.” This is a direct subclass of the StyleSheetControl and has no additional features, messages, or instance data, although the classes are of course internally different. For full information about the StyleSheetControl, see “Generic UI Controllers,” Chapter 12.

18.3.3 GrObjAreaColorSelector

This controller allows the user to set color and fill pattern information for the selected graphic objects.

Code Display 18-2 GrObjAreaColorSelector Features

```
/* GrObjAreaColorSelector is a subclass of ColorSelectorClass and has the same
 * feature set: CSF_FILLED_LIST, CSF_INDEX, CSF_RGB, CSF_DRAW_MASK, CSF_PATTERN).
 */

#define GOACS_DEFAULT_FEATURES (CSF_INDEX | CSF_RGB | CSF_DRAW_MASK )
```

18.3.4 GrObjAreaAttrControl

The GrObjAreaAttrControl allows the user to work with those area attributes which are not handled by the ColorSelector. At this time, all of the features correspond to **MixMode** values which will be used to draw the area.

Code Display 18-3 GrObjAreaAttrControl Features

```
/* GrObjAreaAttrControlClass is a subclass of GenControlClass.
 * Add your GrObjAreaAttrControl to GAGCNLT_SELF_LOAD_OPTIONS. */
```

```
typedef WordFlags GOAACFeatures;
/* The following flags may be combined using | and &:
        GOAACF_MM_CLEAR,
        GOAACF_MM_COPY,
        GOAACF_MM_NOP,
        GOAACF_MM_AND,
        GOAACF_MM_INVERT,
        GOAACF_MM_XOR,
        GOAACF_MM_SET,
        GOAACF_MM_OR,
        GOAACF_TRANSPARENCY */

#define GOAAC_DEFAULT_FEATURES   (GOAACF_TRANSPARENCY | GOAACF_MM_COPY | \
                GOAACF_MM_INVERT | GOAACF_MM_XOR | GOAACF_MM_AND | GOAACF_MM_OR)

#define GOAAC_DEFAULT_TOOLBOX_FEATURES          0
```

18.3.5 GrObjLineColorSelector

The GrObjLineColorSelector is a color selector which has been set up to work with GrObj line attributes. The user may specify a color and mask with which to draw lines.

Code Display 18-4 GrObjLineColorSelector Features

```
/* GrObjLineColorSelector is a subclass of ColorSelectorClass and has the same
 * feature set: CSF_INDEX, CSF_RGB, CSF_DRAW_MASK. Note that since Lines
 * may not draw with a pattern, the CSF_PATTERN feature is not used.
 * Add your GrObjLineColorSelector to GAGCNLT_SELF_LOAD_OPTIONS. */

/* The GOLCS_DEFAULT_FEATURES contains the default feature set */
```

18.3.6 GrObjLineAttrControl

The GrObjLineAttrControl allows the user to specify those attributes of the line not controlled by the GrObjLineColorSelector. This includes line width and style.

Code Display 18-5 GrObjLineAttrControl Features

```
/* GrObjLineAttrControlClass is a subclass of GenControlClass.
 * Add your GrObjLineAttrControl to GAGCNLT_SELF_LOAD_OPTIONS. */

typedef WordFlags GOLACFeatures;
#define GOLACF_WIDTH_INDEX 0x0010
#define GOLACF_WIDTH_VALUE 0x0008
#define GOLACF_STYLE 0x0004
#define GOLACF_ARROWHEAD_TYPE 0x0002
#define GOLACF_ARROWHEAD_WHICH_END 0x0001

typedef WordFlags GOLACToolboxFeatures;
#define GOLACTF_WIDTH_INDEX 0x0002
#define GOLACTF_STYLE 0x0001

#define GOLAC_DEFAULT_FEATURES (GOLACF_WIDTH_INDEX | GOLACF_WIDTH_VALUE | \
                GOLACF_STYLE | GOLACF_ARROWHEAD_TYPE | GOLACF_ARROWHEAD_WHICH_END)

#define GOLAC_DEFAULT_TOOLBOX_FEATURES   (GOLACTF_WIDTH_INDEX | GOLACF_STYLE)
```

18.3.7 GrObjNudgeControl

The GrObjNudgeControl allows the user to “nudge” selected objects by small amounts.

Code Display 18-6 GrObjNudgeControl Features

```
/* GrObjNudgeControlClass is a subclass of GenControlClass.
 * Add your GrObjNudgeControl to GAGCNLT_SELF_LOAD_OPTIONS. */
```

```
typedef ByteFlags GrObjNudgeControlFeatures;
#define GONCF_NUDGE_LEFT        (0x10)
#define GONCF_NUDGE_RIGHT       (0x08)
#define GONCF_NUDGE_UP          (0x04)
#define GONCF_NUDGE_DOWN        (0x02)
#define GONCF_CUSTOM_MOVE       (0x01)

#define GONC_DEFAULT_FEATURES            (GONCF_NUDGE_LEFT | GONCF_NUDGE_RIGHT | \
                        GONCF_NUDGE_UP | GONCF_NUDGE_DOWN | GONCF_CUSTOM_MOVE)
```

18.3.8 GrObjDepthControl

The GrObjDepthControl affects the selected objects' places in the drawing order, so that they will appear higher or lower when they overlap with other objects. Note that since the GrObjBody stores its GrObjs in drawing order, this affects the selected GrObj's place in the drawing order, but other than that does not affect the GrObj itself.

Code Display 18-7 GrObjDepthControl Features

```
/* GrObjDepthControlClass is a subclass of GenControlClass.
 * Add your GrObjDepthControl to GAGCNLT_SELF_LOAD_OPTIONS. */

typedef ByteFlags GODepthCFeatures;
/* The following flags may be combined with | and &:
        GODepthCF_BRING_TO_FRONT,
        GODepthCF_SEND_TO_BACK,
        GODepthCF_SHUFFLE_UP,
        GODepthCF_SHUFFLE_DOWN */

#define GODepthC_DEFAULT_FEATURES \
                        (GODepthCF_BRING_TO_FRONT | GODepthCF_SEND_TO_BACK |\
                         GODepthCF_SHUFFLE_UP | GODepthCF_SHUFFLE_DOWN )
#define GODepthC_DEFAULT_TOOLBOX_FEATURES (\
                        (GODepthCF_BRING_TO_FRONT | GODepthCF_SEND_TO_BACK |\
                         GODepthCF_SHUFFLE_UP | GODepthCF_SHUFFLE_DOWN )
```

Objects

18.3.9 GrObjArcControl

The GrObjArcControl allows the user to specify exact angles and characteristics of arc objects.

Code Display 18-8 GrObjArcControlClass

```
/* GrObjArcControlClass is a subclass of GenControlClass.
 * Add your GrObjDepthControl to GAGCNLT_SELF_LOAD_OPTIONS. */

typedef ByteFlags GOArcCFeatures;
/* The following flags may be combined using | and &:
        GOACF_START_ANGLE,
        GOACF_END_ANGLE,
        GOACF_PIE_TYPE,
        GOACF_CHORD_TYPE */

#define GOArcC_DEFAULT_FEATURES  (GOACF_START_ANGLE | GOACF_END_ANGLE | \
                                  GOACF_PIE_TYPE | GOACF_CHORD_TYPE)

#define GOArcC_DEFAULT_TOOLBOX_FEATURES 0
```

18.3.10 GrObjHandleControl

The GrObjHandleControl allows the user to make the shape's movement and stretching handles smaller, larger, or invisible.

Code Display 18-9 GrObjHandleControl Features

```
/* GrObjHandleControlClass is a subclass of GenControlClass.
 * Add your GrObjHandleControl to GAGCNLT_SELF_LOAD_OPTIONS. */

typedef ByteFlags GOHCFeatures;
/* The following flags may be combined with | and &:
        GOHCF_SMALL_HANDLES,
        GOHCF_MEDIUM_HANDLES,
        GOHCF_LARGE_HANDLES,
        GOHCF_INVISIBLE_HANDLES */
```

```
#define GOHC_DEFAULT_FEATURES (GOHCF_SMALL_HANDLES | GOHCF_MEDIUM_HANDLES |\
                GOHCF_LARGE_HANDLES | GOHCF_INVISIBLE_HANDLES)
#define GOHC_DEFAULTD_TOOLBOX_FEATURES   0
```

18.3.11 GrObjRotateControl

The GrObjRotateControl applies a rotation to the selected GrObj. This normally affect's the object's coordinate transformation.

Code Display 18-10 GrObjRotateControl Features

```
/* GrObjRotateControlClass is a subclass of GenControlClass.
 * Add your GrObjRotateControl to GAGCNLT_SELF_LOAD_OPTIONS. */
typedef ByteFlags GORCFeatures;
typedef ByteFlags GORCFeatures;
#define GORCF_45_DEGREES_CW 0x0080
#define GORCF_90_DEGREES_CW 0x0040
#define GORCF_135_DEGREES_CW 0x0020
#define GORCF_180_DEGREES 0x0010
#define GORCF_135_DEGREES_CCW 0x0008
#define GORCF_90_DEGREES_CCW 0x0004
#define GORCF_45_DEGREES_CCW 0x0002
#define GORCF_CUSTOM_ROTATION 0x0001

#define GORC_DEFAULT_FEATURES 0x00ff
```

18.3.12 GrObjFlipControl

The GrObjFlipControl allows the user to flip a graphic object about either axis.

Code Display 18-11 GrObjFlipControl Features

```
/* GrObjFlipControlClass is a subclass of GenControlClass.
 * Add your GrObjFlipControl to GAGCNLT_SELF_LOAD_OPTIONS. */
```

```
typedef ByteFlags GOFCFeatures;
/* The following flags may be combined with | and &:
        GOFCF_FLIP_HORIZONTALLY,
        GOFCF_FLIP_VERTICALLY */
```

18.3.13 GrObjSkewControl

The GrObjSkewControl provides the user with a standard way to warp graphic objects. This normally affect's the object's coordinate transformation.

Code Display 18-12 GrObjSkewControl Features

```
/* GrObjFlipControlClass is a subclass of GenControlClass.
 * Add your GrObjFlipControl to GAGCNLT_SELF_LOAD_OPTIONS. */

typedef ByteFlags GOSCFeatures;
#define GOSCF_LEFT 0x0010
#define GOSCF_RIGHT 0x0008
#define GOSCF_UP 0x0004
#define GOSCF_DOWN 0x0002
#define GOSCF_CUSTOM_SKEW 0x0001

#define GROBJ_SKEW_CONTROL_DEFAULT_FEATURES 0x001f
```

18.3.14 GrObjAlignToGridControl

The GrObjAlignToGridControl works with a VisRuler's grid mechanism, allowing the user to align selected objects with grid lines. Note that standard ruler controls will allow the user to constrain all mouse movement to grid lines.

Code Display 18-13 GrObjAlignToGridControl Features

```
/* GrObjAlignToGridControlClass is a subclass of GenControlClass.
 * Add your GrObjAlignToGridControl to GAGCNLT_SELF_LOAD_OPTIONS. */
```

```
typedef ByteFlags GOATGCFeatures;
/* There is only one GrObjAlignToGridControl feature:
        GOATGCF_ALIGN_TO_GRID */

#define GOATGC_DEFAULT_FEATURES          (GOATGCF_ALIGN_TO_GRID)
#define GOATGC_DEFAULT_TOOLBOX_FEATURES  0
```

18.3.15 GrObjGroupControl

The GrObjGroupControl allows the user to clump GrObjs together into groups. These GrObjs may then be moved, resized, and skewed in common.

Code Display 18-14 GrObjGroupControl

```
/* GrObjGroupControlClass is a subclass of GenControlClass.
 * Add your GrObjGroupControl to GAGCNLT_SELF_LOAD_OPTIONS. */

typedef ByteFlags GOGCFeatures;
/* These flags may be combined using | and &:
        GOGCF_GROUP,
        GOGCF_UNGROUP */

#define GOGC_DEFAULT_FEATURES            (GOGCF_GROUP | GOGCF_UNGROUP)
#define GOGC_DEFAULT_TOOLBOX_FEATURES    (GOGCF_GROUP | GOGCF_UNGROUP)
```

18.3.16 GrObjAlignDistributeControl

The GrObjAlignDistributeControl allows the user to align several objects so that their edges line up, or to distribute them evenly within a given space.

Objects

Code Display 18-15 GrObjAlignDistrbuteControl Features

```
typedef WordFlags GrObjAlignDistributeControlFeatures;
#define GOADCF_ALIGN_LEFT 0x8000
#define GOADCF_ALIGN_CENTER_HORIZONTALLY 0x4000
#define GOADCF_ALIGN_RIGHT 0x2000
#define GOADCF_ALIGN_WIDTH 0x1000

#define GOADCF_ALIGN_TOP 0x800
#define GOADCF_ALIGN_CENTER_VERTICALLY 0x400
#define GOADCF_ALIGN_BOTTOM 0x200
#define GOADCF_ALIGN_HEIGHT 0x100

#define GOADCF_DISTRIBUTE_LEFT 0x80
#define GOADCF_DISTRIBUTE_CENTER_HORIZONTALLY 0x40
#define GOADCF_DISTRIBUTE_RIGHT 0x20
#define GOADCF_DISTRIBUTE_WIDTH 0x10
#define GOADCF_DISTRIBUTE_TOP 0x8
#define GOADCF_DISTRIBUTE_CENTER_VERTICALLY 0x4
#define GOADCF_DISTRIBUTE_BOTTOM 0x2
#define GOADCF_DISTRIBUTE_HEIGHT 0x1
```

18.3.17 GrObjLocksControl

The GrObjLocksControl allows the user to prohibit certain actions upon a graphic object.

Code Display 18-16 GrObjLocksControl Features

```
/* GrObjLocksControlClass is a subclass of GenControlClass.
 * Add your GrObjLocksControl to GAGCNLT_SELF_LOAD_OPTIONS. */
/* GrObjLocksControl doesn’t have a Features Type; just use the GrObjLocks type. */
```

18.3.18 GrObjConvertControl

The GrObjConvertControl converts an arbitrary collection of graphic objects into a bitmap or GString object.

Code Display 18-17 GrObjConvertControl Features

```
/* GrObjConvertControlClass is a subclass of GenControlClass.
 * Add your GrObjConvertControl to GAGCNLT_SELF_LOAD_OPTIONS. */

typedef ByteFlags GOCCFeatures;
/* The following flags may be combined using | and &:
        GOCCF_CONVERT_TO_BITMAP,
        GOCCF_CONVERT_TO_GRAPHIC,
        GOCCF_CONVERT_FROM_GRAPHIC */

#define GOCC_TOOLBOX_FEATURES (GOCCF_CONVERT_TO_BITMAP | GOCCF_CONVERT_TO_GRAPHIC \
                                | GOCCF_CONVERT_FROM_GRAPHIC)
#define GOCC_TOOLBOX_TOOLBOX_FEATURES (GOCCF_CONVERT_TO_BITMAP | \
                        GOCCF_CONVERT_TO_GRAPHIC | GOCCF_CONVERT_FROM_GRAPHIC)
```

18.3.19 GrObjDefaultAttributesControl

This controller allows the setting of default attributes. It has one feature in its **GODACFeatures** structure: GODACF_SET_DEFAULT_ATTRIBUTES.

18.3.20 GrObjObscureAttrControl

This controller allows the user to specify a number of attributes for each graphic object. The user may change the text wrap flags to determine how text (or other information) should wrap around graphical objects in those applications which support wrapping around graphics. The user can mark an object as an instruction object.

The user may also determine how an object should react depending on the background document, if any. For instance, in spreadsheets, this controller would determine how an object should react if the column it were in were deleted.

Code Display 18-18 GrObjObscureAttrControl Features

```
typedef ByteFlags GrObjObscureAttrControlFeatures;
#define GOOACF_INSTRUCTIONS 0x80
#define GOOACF_INSERT_OR_DELETE_MOVE 0x40
#define GOOACF_INSERT_OR_DELETE_RESIZE 0x20
#define GOOACF_INSERT_OR_DELETE_DELETE 0x10
#define GOOACF_DONT_WRAP 0x08
#define GOOACF_WRAP_INSIDE 0x04
#define GOOACF_WRAP_AROUND_RECT 0x02
#define GOOACF_WRAP_TIGHTLY 0x01

#define GOOAC_INSERT_OR_DELETE_FEATURES (GOOACF_INSERT_OR_DELETE_MOVE | \
                GOOACF_INSERT_OR_DELETE_RESIZE | GOOACF_INSERT_OR_DELETE_DELETE)

#define GOOAC_WRAP_FEATURES (GOOACF_DONT_WRAP | GOOACF_WRAP_INSIDE | \
                                GOOACF_WRAP_AROUND_RECT | GOOACF_WRAP_TIGHTLY)

#define GOOAC_DEFAULT_FEATURES (GOOACF_INSTRUCTIONS | GOOAC_WRAP_FEATURES | \
                                                 GOOAC_INSERT_OR_DELETE_FEATURES)
```

18.3.21 GrObjInstructionControl

This controller allows the user to work with instruction objects. These are objects which have been marked as part of the instructions associated with a document (perhaps so marked using this very controller). These instruction objects might tell the user how to use a template file. This controller allows the user to delete or hide instruction objects. Note that the GrObjObscureAttrControl allows the user to mark an object as an instruction object.

Code Display 18-19 GrObjInstructionControl Features

```
typedef ByteFlags GrObjInstructionControlFeatures;
#define GOICF_DRAW 0x8000
#define GOICF_PRINT 0x4000
#define GOICF_MAKE_EDITABLE 0x2000
#define GOICF_MAKE_UNEDITABLE 0x1000
#define GOICF_DELETE 0x0800

#define GOICF_DEFAULT_FEATURES (GOICF_DRAW | GOICF_PRINT | GOICF_MAKE_EDITABLE | \
                                        GOICF_MAKE_UNEDITABLE | GOICF_DELETE)
```

18.3.22 GrObjGradientFillControl

This controller allows the user to choose the specifics of a gradient fill, including starting and ending colors, direction of fill, and more.

18.3.23 GrObjBackgroundColorSelector

This controller allows the user to specify the background color of a graphic object, useful when the object is partially see-through.

18.3.24 Gradient Color Selectors

The **GrObjStartingGradientColorSelector** and **GrObjEndingGradientColorSelector** controllers allow the user to specify both ends of a gradient-filled object's color range.

18.3.25 Paste Inside Controls

A **GrObjPasteInsideControl** allows the user to paste one object inside of another, allowing for complicated clipping effects. The

◆Objects

GrObjMoveInsideControl controller allows the "nudging" of an object which has been pasted inside of another object.

Code Display 18-20 GrObjPasteInsideControl Features

```
typedef ByteFlags GOPICFeatures;
#define GOPICF_PASTE_INSIDE 0x0002
#define GOPICF_BREAKOUT_PASTE_INSIDE 0x0001

typedef ByteFlags GOPICToolboxFeatures;
#define GOPICTF_PASTE_INSIDE 0x0002
#define GOPICTF_BREAKOUT_PASTE_INSIDE 0x0001

#define GOPIC_DEFAULT_FEATURES (GOPICF_PASTE_INSIDE | GOPICF_BREAKOUT_PASTE_INSIDE)

#define GOPIC_DEFAULT_TOOLBOX_FEATURES (GOPICTF_PASTE_INSIDE | \
                                                GOPICTF_BREAKOUT_PASTE_INSIDE)
```

18.3.26 Controls From Other Libraries

Because the grobj incorporates the powers of many other libraries, applications working with the grobj may have cause to include controllers from some of these other libraries.

The text library includes several controllers, which can work with GrObj text objects as they would with VisText objects.

If your geode includes one or more VisRulers, including some ruler controls will allow the user to customize some ruler behavior. These ruler controllers are discussed in some detail in the "Ruler Object Library," Chapter 19.

The bitmap library provides controls useful for manipulating bitmaps under the GrObj. Though the bitmap library is internal to Geoworks, the GrObj library provides the VisBitmapToolControl which will allow the user to manipulate bitmaps via GrObj.

18.4 GrObj Body

The GrObjBody represents a layer of graphics. Most of your application's interaction with the world of GrObj will probably be through the body.

18.4.1 GrObjBody Instance Data

The GrObjBody largely concerns itself with managing other objects. Much of its instance data is taken up with handles of the various objects that it interacts with.

Code Display 18-21 GrObjBody Instance Data

```
@instance RectDWord     GBI_bounds =             {0,0,0,0};
@instance CompPart      GBI_drawComp =           {NullOptr};     /* Internal */
@instance CompPart      GBI_reverseComp =        {NullOptr};     /* Internal */
@instance word          GBI_childCount;                          /* Internal */
@instance optr          GBI_selectionArray;                      /* Internal */
@instance HierarchicalGrab GBI_targetExcl =      {NullOptr, 0};  /* Internal */
@instance HierarchicalGrab GBI_focusExcl =       {NullOptr, 0};  /* Internal */
@instance BasicGrab     GBI_curEdit =            {NullOptr, 0};  /* Internal */
@instance optr          GBI_mouseGrab;                           /* Internal */
@instance word          GBI_objBlockArray;                       /* Internal;*/
@instance GrObjFunctionsActive GBI_defaultOptions;
@instance GrObjFunctionsActive GBI_currentModifiers;
@instance GrObjFunctionsActive GBI_currentOptions;
@instance GrObjBodyFlags GBI_flags = (GBF_DEFAULT_TARGET | GBF_DEFUALT_FOCUS);
@instance GrObjDrawFlags GBI_drawFlags;
@instance GrObjFileStatus GBI_fileStatus;                        /* Internal */
@instance GStateHandle  GBI_graphicsState =      0;              /* Internal */
@instance optr          GBI_head;
@instance optr          GBI_goam;
@instance optr          GBI_ruler;
@instance word          GBI_priorityList =       0;              /* Internal */
@instance byte          GBI_desiredHandleSize =  DEFAULT_DESIRED_HANDLE_SIZE;
@instance byte          GBI_curHandleWidth =     0;              /* Internal */
@instance byte          GBI_curHandleHeight =    0;              /* Internal */
@instance BBFixed       GBI_curNudgeX;                           /* Internal */
@instance BBFixed       GBI_curNudgeY;                           /* Internal */
@instance PointWWFixed  GBI_curScaleFactor = {MakeWWFixed(1), MakeWWFixed(0)};
```

```
@instance PointDWFixed  GBI_interestingPoint = {{0, -30000}, {0, -30000}};
@instance PointDWFixed  GBI_lastPtr = {0,0};
@instance word          GBI_suspendCount =      0;              /* Internal */
@instance GrObjBodyUnsuspendOps GBI_unsuspendOps;               /* Internal */
@instance VisTextNotificationFlags GBI_textUnsuspendOps = 0;    /* Internal */
@instance word GBI_reserved1 = 0;               /* Reserved for future use */
@instance word GBI_reserved2 = 0;               /* Reserved for future use */

@vardata GrObjActionNotification ATTR_GB_ACTION_NOTIFICATION;

@vardata GrObjBodyPasteCallBackStruct ATTR_GB_PASTE_CALL_BACK;
typedef struct {
 word GOBPCBS_message;
 optr GOBPCBS_optr;
} GrObjBodyPasteCallBackStruct;
```

The *GBI_bounds* field acts as the bounds of the drawing area. Geodes using graphic bodies as part of documents may wish to make the body's bounds match the document bounds. Use MSG_GB_SET_BOUNDS to reset these bounds.

When working with a graphic layer, the user will no doubt have certain preferences about how input should be interpreted. These preferences include the option to "stretch" objects from the center, snapping to the grid, and drawing bitmaps as plain rectangles. One set of options may be set as the default, stored in *GBI_defaultOptions*. As input events come in, the body may turn on other options, which are stored in *GBI_currentModifiers*. The *GBI_currentOptions* field contains the computed combination of these options.

```
typedef WordFlags GrObjFunctionsActive;
/* These flags may be combined with | and &:
      GOFA_HAS_SEEN_EVENT,
      GOFA_VIEW_ZOOMED,
      GOFA_SNAP_TO,
      GOFA_FROM_CENTER,
      GOFA_ABOUT_OPPOSITE,
      GOFA_CONSTRAIN,
      GOFA_ADJUST,
      GOFA_EXTEND */
```

The *GBI_flags* field determines some of the GrObjBody's miscellaneous behavior.

```
typedef WordFlags GrObjBodyFlags;
/* These flags may be combined with | and &:
      GBF_HAS_ACTION_NOTIFICATION,
      GBF_DEFAULT_TARGET,
      GBF_DEFAULT_FOCUS */
```

The *GBI_goam* field is the handle of the GrObjBody's attribute manager. The *GBI_head* field is the handle of the GrObjHead. The *GBI_ruler* field may hold a handle to a VisRuler. If used, this ruler will be able to coordinate with the GrObjBody to provide mouse tracking.

18.4.2 GrObjBody Messages

Since the body is in charge of managing the "shape" objects, several of these messages deal with adding and removing shapes from the selection and shuffling objects up and down in the drawing order.

■ MSG_GB_ATTACH_UI

```
void    MSG_GB_ATTACH_UI (
        optr   GrObjHead);
```

This message lets the GrObjBody know where its head is. The head's OD will be set in the instance data. This message must be sent to the GrObjBody after it has been added to a document.

Source: Unrestricted, normally a GenDocument handling MSG_GEN_DOCUMENT_ATTACH_UI.

Destination: GrObjBody.

Parameters: *head* The optr of the GrObjHead to use.

Return: Nothing.

Interception: Unlikely.

■ MSG_GB_DETACH_UI

```
void      MSG_GB_DETACH_UI ();
```

This message must be sent to the GrObjBody before it has been removed to a document, and should be in the document's MSG_GEN_DOCUMENT_ATTACH_UI.

Source: Unrestricted, normally a GenDocument handling MSG_GEN_DOCUMENT_DETACH_UI.

Destination: GrObjBody.

Parameters: None.

Return: Nothing.

Interception: Unlikely.

■ MSG_GB_ATTACH_GOAM

```
void      MSG_GB_ATTACH_GOAM (
          optr   GrObjAttrManager);
```

This message lets the GrObjBody know where its attribute manager is. The manager's optr will be set in the instance data.

Source: Unrestricted. Generally the GenDocument on handling MSG_GEN_DOCMENT_INITIALIZE_DOCUMENT_FILE.

Destination: GrObjBody.

Parameters: *goam* The optr of the new GrObjAttributeManager.

Return: Nothing.

Interception: Unlikely.

■ MSG_GB_ATTACH_RULER

This message sets the *GBI_ruler* field. Note that if the GrObjBody is to interact with more than one VisRuler, they should be linked in a *VRI_slaveRuler* chain, and the ruler in the *GBI_ruler* field should be at the top of this chain.

Source: Unrestricted. Generally the GenDocument on handling MSG_GEN_DOCMENT_INITIALIZE_DOCUMENT_FILE.

Destination: GrObjBody.

Parameters: *ruler* The optr of the new VisRuler to use.

Return: Nothing.

Interception: Unlikely.

■ MSG_GB_ADD_GROBJ

```
void	MSG_GB_ADD_GROBJ (
	optr	object,
	word	flags);
```

This message adds a graphic object to the GrObjBody. The object will be notified by a MSG_GO_AFTER_ADDED_TO_BODY that it has been added to the body. The object won't necessarily be drawn; to force a redraw, send MSG_GB_ADD_GROBJ_THEN_DRAW instead.

Source: Unrestricted. Generally the GenDocument on handling MSG_GEN_DOCMENT_INITIALIZE_DOCUMENT_FILE.

Destination: GrObjBody.

Parameters: *object* The optr of the new GrObj.

flags **GrObjBodyAddGrObjFlags** value specifying where in the list of objects to add the new GrObj.

Return: Nothing.

Structures: The flags field is a word with the following structure:

```
typedef WordFlags GrObjBodyAddGrObjFlags;
      /* GOBAGOF_DRAW_LIST_POSITION: if this bit
            is set, then other bits describe
            position in drawing-order list. If
            this bit is clear, then other bits
            describe position in reverse list */
      #define GOBAGOF_DRAW_LIST_POSITION 0x8000
      #define GOBAGOF_REFERENCE 0x7fff
      #define GOBAGOR_FIRST CCO_FIRST
      #define GOBAGOR_LAST CCO_LAST
/* To add a new object so it draws “on top”, use
      (GOBAGOF_DRAW_LIST_POSITION | GOBAGOR_LAST) */
```

Interception: Unlikely.

◆ Objects

■ MSG_GB_ADD_GROBJ_THEN_DRAW

```
void        MSG_GB_ADD_GROBJ_THEN_DRAW(
            optr   object,
            word flags);
```

This message adds a graphic object to the GrObjBody. The object will be notified by a MSG_GO_AFTER_ADDED_TO_BODY that it has been added to the body. If the object is added at the top of the draw list, it will be sent a draw message. Otherwise, it will be invalidated.

Source: Unrestricted. Generally the GenDocument on handling MSG_GEN_DOCMENT_INITIALIZE_DOCUMENT_FILE.

Destination: GrObjBody.

Parameters:

object	The optr of the new GrObj.
flags	Object, specific flags, if any.

Return: Nothing.

Structures: The flags field is a word with the following structure:

```
typedef WordFlags GrObjBodyAddGrObjFlags;
      /* GOBAGOF_DRAW_LIST_POSITION: if this bit
            is set, then other bits describe
            position in drawing-order list. If
            this bit is clear, then other bits
            describe position in reverse list */
      #define GOBAGOF_DRAW_LIST_POSITION 0x8000
      #define GOBAGOF_REFERENCE 0x7fff
      #define GOBAGOR_FIRST CCO_FIRST
      #define GOBAGOR_LAST CCO_LAST
/* To add a new object so it draws “on top”, use
      (GOBAGOF_DRAW_LIST_POSITION | GOBAGOR_LAST) */
```

Interception: Unlikely.

■ MSG_GB_SET_BOUNDS

```
void        MSG_GB_SET_BOUNDS(
            RectDWord bounds);
```

This message sets the bounds of the GrObjBody.

Source: Unrestricted.

Destination: GrObjBody.

Parameters: *bounds* The new bounds.

Return: Nothing.

Interception: Unlikely.

■ MSG_GB_INSTANTIATE_GROBJ

```
optr    MSG_GB_INSTANTIATE_GROBJ(
        ClassStruct *class);
```

This message instantiates a GrObj of the passed class in a block managed by the body.

Source: Unrestricted.

Destination: GrObjBody.

Parameters: *bounds* The new bounds.

Return: Object pointer of the newly created GrObj.

Interception: Unlikely.

■ MSG_GB_SET_ACTION_NOTIFICATION_OUTPUT

This message specifies the default message and OD for GrObjs to send notification to when an action is performed on them. GrObjs will use this default if they don't have their own action notification.

Source: Unrestricted.

Destination: GrObjBody.

Interception: Unlikely.

■ MSG_GB_SUSPEND_ACTION_NOTIFICATION

```
void    MSG_GB_SUSPEND_ACTION_NOTIFICATION();
```

This message suspends action notification for all of a body's GrObjs. This prevents all the GrObjs from sending out any action notification.

Source: Unrestricted.

Destination: GrObjBody.

Parameters: None.

Return: Nothing.

Interception: Unlikely.

◆ Objects

■ MSG_GB_UNSUSPEND_ACTION_NOTIFICATION

```
void        MSG_GB_UNSUSPEND_ACTION_NOTIFICATION();
```

This message counteracts MSG_GB_SUSPEND_ACTION_NOTIFICATION. If all suspends have been balanced, the body's GrObjs will be free to send out action notification. However, action notifications that were aborted during the suspend period will not be sent out.

Source: Unrestricted.

Destination: GrObjBody.

Parameters: None.

Return: Nothing.

Interception: Unlikely.

■ MSG_GB_DRAW

```
void        MSG_GB_DRAW(
            GStateHandle        gstate,
            DrawFlags           visDrawFlags,
            GrObjDrawFlags      GODrawFlags);
```

This message draws the graphic layer, allowing the caller some options. Since the GrObjBody is a subclass of VisComp, of course it is normally drawn by MSG_VIS_DRAW. MSG_GB_DRAW is normally used directly when drawing the layer to something not normally its output.

Source: Unrestricted.

Destination: GrObjBody.

Parameters:

gstate	The graphics state to draw to.
visDrawFlags	Flags as would be set for MSG_VIS_DRAW.
GODrawFlags	Special flags which can affect how to draw GrObjs.

Return: Nothing.

Structures:

```
typedef ByteFlags GrObjDrawFlags;
#define GODF_DRAW_QUICK_VIEW 0x100
#define GODF_DRAW_CLIP_ONLY 0x80
#define GODF_DRAW_WRAP_TEXT_INSIDE_ONLY 0x40
#define GODF_DRAW_WRAP_TEXT_AROUND_ONLY 0x20
#define GODF_DRAW_WITH_INCREASED_RESOLUTION 0x10
```

```
#define GODF_DRAW_INSTRUCTIONS 0x08
#define GODF_DRAW_SELECTED_OBJECTS_ONLY 0x04
#define GODF_DRAW_OBJECTS_ONLY 0x02
#define GODF_PRINT_INSTRUCTIONS 0x01
```

Interception: Unlikely.

■ MSG_GB_GIVE_ME_MOUSE_EVENTS

18.4 ■ MSG_GB_DONT_GIVE_ME_MOUSE_EVENTS

■ MSG_GB_FIND_GROBJ

■ MSG_GB_PULL_SELECTED_GROBJS_TO_FRONT

■ MSG_GB_PUSH_SELECTED_GROBJS_TO_BACK

■ MSG_GB_SHUFFLE_SELECTED_GROBJS_UP

■ MSG_GB_SHUFFLE_SELECTED_GROBJS_DOWN

■ MSG_GB_SET_DESIRED_HANDLE_SIZE

```
void     MSG_GB_SET_DESIRED_HANDLE_SIZE(
         byte   handleSize);
```

■ **MSG_GB_REMOVE_GROBJ**

■ **MSG_GB_ADD_GROBJ_TO_SELECTION_LIST**

■ **MSG_GB_MESSAGE_TO_FLOATER_IF_PARENT**

■ **MSG_GB_UPDATE_UI_CONTROLLERS**

■ **MSG_GB_ADD_DUPLICATE_FLOATER**

■ **MSG_GB_PRIORITY_LIST_RESET**

■ **MSG_GB_PRIORITY_LIST_GET_NUM_ELEMENTS**

■ **MSG_GB_PRIORITY_LIST_GET_ELEMENT**

■ **MSG_GB_PRIORITY_LIST_INIT**

■ **MSG_GB_GROUP_SELECTED_GROBJS**

■ **MSG_GB_UNGROUP_SELECTED_GROUPS**

■ **MSG_GB_TRANSFER_GROBJ_FROM_GROUP**

■ **MSG_GB_CLOSE_FINISH_UP**

■ **MSG_GB_CLEAR**

■ **MSG_GB_ALIGN_SELECTED_GROBJS**

■ **MSG_GB_CREATE_SORTABLE_ARRAY**

■ **MSG_GB_DESTROY_SORTABLE_ARRAY**

■ **MSG_GB_FILL_SORTABLE_ARRAY_USING_GO_DWF_BOUNDS**

■ **MSG_GB_FILL_SORTABLE_ARRAY_USING_GO_CENTERS**

■ **MSG_GB_SORT_SORTABLE_ARRAY**

■ **MSG_GB_GET_CENTER_OF_SELECTED_GROBJS**

■ **MSG_GB_GET_CENTER_OF_FIRST_SELECTED_GROBJ**

■ MSG_GB_GET_CENTER_OF_LAST_SELECTED_GROBJ

■ MSG_GB_GET_DWF_BOUNDS_OF_FIRST_SELECTED_GROBJ

■ MSG_GB_GET_DWF_BOUNDS_OF_LAST_SELECTED_GROBJ

■ MSG_GB_GET_WINDOW

18.4

■ MSG_GB_GET_BOUNDS

```
void      MSG_GB_GET_BOUNDS(
          RectDWord *bounds);
```

■ MSG_GB_SUBST_AREA_TOKEN

■ MSG_GB_SUBST_LINE_TOKEN

■ MSG_GB_CREATE_GROBJ_TRANSFER_FORMAT

■ MSG_GB_CREATE_GSTRING_TRANSFER_FORMAT

■ MSG_GB_IMPORT

■ MSG_GB_EXPORT

■ MSG_GB_GRAB_TARGET_FOCUS

■ MSG_GB_GENERATE_TEXT_NOTIFY

■ MSG_GB_GENERATE_SPLINE_NOTIFY

■ MSG_GB_DETACH_GOAM

■ MSG_GB_SEND_CLASSED_EVENT_SET_DEFAULT_ATTRS

■ MSG_GB_REMOVE_GROBJ_FROM_SELECTION_LIST

■ MSG_GB_REMOVE_ALL_GROBJS_FROM_SELECTION_LIST

■ MSG_GB_GET_NUM_SELECTED_GROBJS

■ MSG_GB_GET_BOUNDS_OF_SELECTED_GROBJS

■ MSG_GB_GET_DWF_BOUNDS_OF_SELECTED_GROBJS

◆Objects

■ **MSG_GB_GET_SUMMED_DWF_DIMENSIONS_OF_SELECTED_GROBJS**

■ **MSG_GB_DELETE_SELECTED_GROBJS**

■ **MSG_GB_PROCESS_ALL_GROBJS_IN_RECT**

■ **MSG_GB_INCREASE_POTENTIAL_EXPANSION**

■ **MSG_GB_DECREASE_POTENTIAL_EXPANSION**

■ **MSG_GB_INVALIDATE**

■ **MSG_GB_CONVERT_SELECTED_GROBJS_TO_BITMAP**

■ **MSG_GB_CONVERT_SELECTED_GROBJS_TO_GRAPHIC**

■ **MSG_GB_CREATE_GSTATE**

18.5 GrObjHead

The GrObjHead, as previously mentioned, is in charge of maintaining tool information for an entire application. Theoretically, you could have more than one GrObjHead for an application, but this would lead to confusion for the user, and could easily become confusing for you as well.

When declaring the GrObjHead, you may give a value for the *GH_currentTool* field. For instance, by passing **RectClass** in this field, the head will automatically start out set up ready for the user to draw rectangles. This is an especially useful option which only allows the user to work with one sort of object, and will thus not include a tool selector control.

Code Display 18-22 GrObjHead Instance Data

```
@instance ClassStruct   *GH_currentTool = NullClass;
@instance word          GH_initializeFloaterData = 0;           /* Internal */
@instance optr          GH_currentBody;
@instance optr          GH_floater;
```

18.5 GrObjHead messages are, concerned with the current tool.

■ MSG_GH_GET_CURRENT_TOOL

```
void    MSG_GH_GET_CURRENT_TOOL (
        CurrentToolValues *retVal);
```

This message returns the value of the currently active tool.

Source: Unrestricted.

Destination: GrObjHead.

Parameters: *retVal* Pointer to **CurrentToolValues**, which will hold return value.

Return: Nothing is returned explicitly.

retVal Pointer to **CurrentToolValues** structure.

Structures: The **CurrentToolValues** structure is defined as follows:

```
typedef struct{
      word              CTV_grObjSpecificData;
      word              CTV_unused;
      ClassStruct       *CTV_toolClass;
} CurrentToolValues;
```

Interception: Unlikely.

■ MSG_GH_SET_CURRENT_TOOL

```
void    MSG_GH_SET_CURRENT_TOOL (
        ClassStruct         *class,
        word                initData);
```

This message activates a tool.

Source: Unrestricted.

Destination: GrObjHead.

Parameters: *class* Class of the new tool.

initData Tool class-specific initialization data.

Return: Nothing.

Interception: Unlikely.

■ MSG_GH_SET_CURRENT_BODY

■ MSG_GH_CLEAR_CURRENT_BODY

■ MSG_GH_CLASSED_EVENT_TO_FLOATER

■ MSG_GH_CLASSED_EVENT_TO_FLOATER_IF_CURRENT_BODY

■ MSG_GH_FLOATER_FINISHED_CREATE

■ MSG_GH_SEND_NOTIFY_CURRENT_TOOL

■ MSG_GH_SET_CURRENT_TOOL_WITH_DATA_BLOCK

```
void MSG_GH_SET_CURRENT_TOOL_WITH_DATA_BLOCK(
        ClassStruct         *toolClass,
        word                initData); /* Handle of initialization block */
```

18.6 GrObjAttributeManager

The GrObjAttributeManager keeps track of drawing properties to use, maintaining arrays of line, area, and text drawing attributes.

When the user changes the drawing properties of an object, the GOAM must maintain its data structures. It checks to see whether that object was the only one to use its old drawing properties. If it was, then the GOAM frees the item used to store those properties. Then it checks the new properties. If it already has an item corresponding to these properties, it notes that another object is using the property item. Otherwise, it creates a new item.

The GrObjAttributeManager's messages are concerned with maintaining and updating the attribute data structures.

■ MSG_GOAM_CREATE_ALL_ARRAYS

```
void	MSG_GOAM_CREATE_ALL_ARRAYS();
```

This message initializes the data structures in which the GOAM will store all attribute information.

Source: Unrestricted, often a MSG_GEN_DOCUMENT_INITIALIZE_DOCUMENT_FILE handler.

Destination: GrObjAttributeManager.

Return: Nothing.

Interception: Unlikely.

Objects

■ **MSG_GOAM_ATTACH_AND_CREATE_ARRAYS**

■ **MSG_GOAM_ADD_AREA_ATTR_ELEMENT**

■ **MSG_GOAM_ADD_LINE_ATTR_ELEMENT**

■ **MSG_GOAM_DEREF_AREA_ATTR_ELEMENT_TOKEN**

■ **MSG_GOAM_DEREF_LINE_ATTR_ELEMENT_TOKEN**

■ **MSG_GOAM_ADD_REF_AREA_ATTR_ELEMENT_TOKEN**

■ **MSG_GOAM_ADD_REF_LINE_ATTR_ELEMENT_TOKEN**

■ **MSG_GOAM_GET_FULL_AREA_ATTR_ELEMENT**

■ **MSG_GOAM_GET_FULL_LINE_ATTR_ELEMENT**

■ **MSG_GOAM_GET_STYLE_ARRAY**

■ **MSG_GOAM_GET_AREA_ATTR_ARRAY**

■ **MSG_GOAM_GET_LINE_ATTR_ARRAY**

■ **MSG_GOAM_ATTACH_BODY**

■ **MSG_GOAM_INVALIDATE_BODIES**

■ **MSG_GOAM_SUBST_AREA_TOKEN**

■ **MSG_GOAM_SUBST_LINE_TOKEN**

■ **MSG_GOAM_GET_TEXT_OD**

■ **MSG_GOAM_GET_TEXT_ARRAYS**

■ **MSG_GOAM_LOAD_STYLE_SHEET_PARAMS**

■ **MSG_GOAM_DETACH_BODY**

18.7 Graphic Objects

Members of GrObjClass act as the shape objects which appear in a graphics layer. Most applications don't concern themselves with graphic objects, and just let the GrObjBody manage them. As a general rule, the only applications which will work directly with graphic objects are those which create their own graphic objects and add them to a layer.

18.7.1 GrObj Instance Data

Most applications will never work with GrObj instance data in any way. Some applications may work with the *GOI_locks* field. This field acts to disallow performing certain operations on the object. Thus, if your application provides an object which the user should not be able to resize, then set the GOL_RESIZE lock.

```
typedef WordFlags GrObjLocks;
/* These fields may be combined with | and &:
      GOL_COPY,
      GOL_LOCK,
      GOL_SHOW,
      GOL_WRAP,
      GOL_MOVE,
      GOL_RESIZE,
      GOL_ROTATE,
      GOL_SKEW,
      GOL_EDIT,
      GOL_DELETE,
      GOL_SELECT,
      GOL_ATTRIBUTE,
      GOL_GROUP,
      GOL_UNGROUP,
      GOL_DRAW,
      GOL_PRINT,
Remember, these are locks: if the bit is SET,
then the action is FORBIDDEN! */
```

Code Display 18-23 GrObjClass Instance Data

```
@instance LinkPart      GOI_drawLink;                           /* Internal */
@instance LinkPart      GOI_reverseLink;                        /* Internal */
@instance GrObjAttrFlags        GOI_attrFlags = (GOAF_INSERT_DELETE_MOVE_ALLOWED |
        GOAF_INSERT_DELETE_RESIZE_ALLOWED | GOAF_INSERT_DELETE_DELETE_ALLOWED );
/*
 *      typedef WordFlags GrObjAttrFlags;
 *      #define GOAF_DONT_COPY_LOCKS 0x0200
 *      #define GOAF_HAS_PASTE_INSIDE_CHILDREN 0x0100
 *      #define GOAF_PASTE_INSIDE 0x0080
 *      #define GOAF_INSERT_DELETE_MOVE_ALLOWED 0x0040
 *      #define GOAF_INSERT_DELETE_RESIZE_ALLOWED 0x0020
 *      #define GOAF_INSERT_DELETE_DELETE_ALLOWED 0x0010
 *      #define GOAF_INSTRUCTION 0x0008
 *      #define GOAF_MULTIPLICATIVE_RESIZE 0x0004
 *      #define GOAF_WRAP 0x0003
 */

@instance LinkPart GOI_drawLink;                                /* Internal */

@instance LinkPart GOI_reverseLink;                             /* Internal */

@instance GrObjAttrFlags GOI_attrFlags = (GOAF_INSERT_DELETE_MOVE_ALLOWED | \
        GOAF_INSERT_DELETE_RESIZE_ALLOWED | GOAF_INSERT_DELETE_DELETE_ALLOWED );
                                                                /* Internal */

@instance GrObjOptimizationFlags GOI_optFlags =(GOOF_GROBJ_INVALID); /* Internal */
@instance GrObjMessageOptimizationFlags GOI_msgOptFlags =(0);   /* Internal */
@instance GrObjLocks    GOI_locks =             0;
@instance GrObjActionModes GOI_actionModes =    0;              /* Internal */
@instance GrObjTempModes GOI_tempState =        0;              /* Internal */
@instance ChunkHandle   GOI_normalTransform =   NullChunk;      /* Internal */
@instance ChunkHandle   GOI_spriteTransform =   NullChunk;      /* Internal */
@instance word GOI_areaAttrToken = CA_NULL_ELEMENT;             /* Internal */
@instance word GOI_lineAttrToken = CA_NULL_ELEMENT;             /* Internal */
```

```
@vardata GrObjActionNotificationStruct ATTR_GO_ACTION_NOTIFICATION;
@vardata PointWWFixed ATTR_GO_PARENT_DIMENSIONS_OFFSET;          /* Internal */
```

18.7.2 GrObj Messages

Most applications will never send any messages to a graphic object. For the most part, graphic controllers and the Graphic Body tell GrObjs everything that they need to know. It is possible to send a GrObj message to all selected GrObjs by sending it as a classed event to the graphic body, which will relay the message correctly.

The only applications which normally send any of these messages directly to a graphic object are those which instantiate a graphic object and need to initialize its data.

18.7.2.1 Creation and Destruction

■ MSG_GO_INITIALIZE

```
void    MSG_GO_INITIALIZE(
        GrObjInitializeData *data)
```

This message initializes the object's size and position. It also causes the object to take on the default drawing attributes and to do any other initialization necessary before the object is added to the GrObjBody.

Source: Unrestricted.

Destination: New GrObj.

Parameters: *data* The new GrObj's size and position.

Return: Nothing.

Structures:

```
typedef struct {
      PointDWFixed      GOID_position;
      WWFixed           GOID_width;
      WWFixed           GOID_height;
} GrObjInitializeData
```

Objects

Interception: Possible. Objects with additional instance data should subclass this message.

■ MSG_GO_NOTIFY_GROBJ_VALID

```
void        MSG_GO_NOTIFY_GROBJ_VALID();
```

This message notifies the GrObj that it's ready for action. The GrObj has all attributes that it needs to draw, including a transformation. This message is sent to the object at the end of an interactive create or after it has been created statically.

Source: For interactive creates, the object will send this message to itself. If statically created, whatever created the object must make sure this message is then sent to the object.

Destination: GrObj.

Parameters: None.

Return: Nothing.

Interception: Possible, but unlikely.

■ MSG_GO_AFTER_ADDED_TO_BODY

```
void        MSG_GO_AFTER_ADDED_TO_BODY ();
```

This message is sent to an object just after it has been added to a GrObjBody.

Source: GrObjBody only.

Destination: GrObj.

Parameters: None.

Return: Nothing.

Interception: Possible. Call superclass before doing own processing.

■ MSG_GO_BEFORE_REMOVED_FROM_BODY

```
void        MSG_GO_BEFORE_REMOVED_FROM_BODY ();
```

This message is sent to an object just before it is removed from a GrObjBody.

Source: GrObjBody only.

Destination: GrObj.

Parameters: None.

Return: Nothing.

Interception: Possible. Call superclass before doing own processing.

18.7.2.2 Locking Actions

■ MSG_GO_CHANGE_LOCKS

```
dword     MSG_GO_CHANGE_LOCKS (
          GrObjLocks setBits,
          GrObjLocks clearBits);
```

18.7

This message changes the locks on an object. The object's instance data will be changed accordingly.

Source: Unrestricted.

Destination: GrObj.

Parameters:

setBits	Locks to activate, representing forbidden actions.
clearBits	Locks to turn off, actions will be allowed. If GOL_LOCK is set, then this message will change the object's locks even if the GrObj's GOL_LOCK bit had been previously set.

Return: The returned value is a dword formed by concatenating two words. The high word is the **GrObjLocks** value of the object's locks before the change. The low word is the **GrObjLocks** value after the change. Applications which are making temporary locks may wish to save the old values. Mathematically, the return value is (old values << 16) + new values.

Interception: Unlikely.

18.7.2.3 Drawing Attributes

■ MSG_GO_SET_AREA_ATTR

```
void      MSG_GO_SET_AREA_ATTR(
          GrObjBaseAreaAttrElement *_far *attr);
```

This message sets the area attributes of the object.

Source: Unrestricted.

Destination: GrObj.

Parameters:

AreaAttrElement	The attributes to use.

Return: Nothing.

Structures:

```
typedef struct {
      StyleSheetElementHeader GOBAAE_styleElement;
      byte              GOBAAE_r;
      byte              GOBAAE_g;
      byte              GOBAAE_b;
      SysDrawMask       GOBAAE_mask;
      MixMode           GOBAAE_drawMode;
      GraphicPattern    GOBAEE_pattern;
      byte              GOBAEE_backR;
      byte              GOBAEE_backG;
      byte              GOBAEE_backB;
      GrObjAreaAttrElementType GOBAEE_aaeType;
      GrObjAreaAttrInfoRecord GOBAAE_areaInfo;

/* The following fields are unused, but must
 * be initialized to zero. */
      byte              GOBAAE_reservedByte;
      word              GOBAAE_reserved;
} GrObjBaseAreaAttrElement;
```

```
typedef enum {
      GOAAET_BASE,
      GOAAET_GRADIENT
} GrObjAreaAttrElementType;
```

```
typedef ByteFlags GrObjAreaAttrInfoRecord;
#define GOAAIR_TRANSPARENCY 0x80
```

Interception: Unlikely.

■ MSG_GO_SET_AREA_COLOR

```
void    MSG_GO_SET_AREA_COLOR(
        byte   red,
        byte   green,
        byte   blue);
```

This message sets a GrObj's area color.

Source: Unrestricted.

Destination: GrObj.

Parameters: *red* The color's red component.

green The color's green component.

blue The color's blue component.

Return: Nothing.

Interception: Unlikely.

■ MSG_GO_SET_AREA_MASK

18.7

```
void    MSG_GO_SET_AREA_MASK(
        SysDrawMask         mask);
```

This message sets a GrObj's area mask.

Source: Unrestricted.

Destination: GrObj.

Parameters: *mask* The new mask.

Return: Nothing.

Interception: Unlikely.

■ MSG_GO_SET_AREA_DRAW_MODE

```
void    MSG_GO_SET_AREA_DRAW_MODE(
        MixMode             mode);
```

This message sets a GrObj's area mix mode.

Source: Unrestricted.

Destination: GrObj.

Parameters: *mode* The new mix mode.

Return: Nothing.

Interception: Unlikely.

■ MSG_GO_SET_AREA_INFO

```
void    MSG_GO_SET_TRANSPARENCY(
        byte   transparent);
```

This message sets a GrObj's area information flags, so that the object may have a transparent area.

Source: Unrestricted.

Destination: GrObj.

◆ Objects

Parameters: *transparent* Byte value: non-zero for transparent.

Return: Nothing.

Structures: The **AreaAttrInfoRecord** has one flag: AAIR_TRANSPARENT.

Interception: Unlikely.

■ MSG_GO_SET_LINE_ATTR

```
void        MSG_GO_SET_LINE_ATTR(
            GrObjBaseLineAttrElement *attr);
```

This message sets a GrObj's line attributes.

Source: Unrestricted.

Destination: GrObj.

Parameters: *attr* The new line attributes.

Return: Nothing.

Structures:

```
typedef struct {
      StyleSheetElementHeader
                        GOBGOBLAE_styleElement;
      byte              GOBLAE_r;
      byte              GOBLAE_g;
      byte              GOBLAE_b;
      LineEnd           GOBLAE_end;
      LineJoin          GOBLAE_join;
      WWFixed           GOBLAE_width;
      SystemDrawMask    GOBLAE_mask;
      LineStyle         GOBLAE_style;
      WWFixed           GOBLAE_miterLimit;
      GrObjLineAttrElementType GOBLAE_laeType;
      GrObjLineAttrInfoRecord GOBLAE_lineInfo;
      byte              GOBLAE_arrowheadAngle;
      byte              GOBLAE_arrowheadLength;
      word              GOBLAE_reserved;
} GrObjBaseLineAttrElement;
typedef enum {
      GOLAET_BASE
} GrObjLineAttrElementType;
```

```
typedef ByteFlags GrObjLineAttrInfoRecord;
#define GOLAIR_ARROWHEAD_ON_START 0x80
#define GOLAIR_ARROWHEAD_ON_END 0x40
#define GOLAIR_ARROWHEAD_FILLED 0x20
#define GOLAIR_ARROWHEAD_FILL_WITH_AREA_ATTRIBUTES 0x10
```

Interception: Unlikely.

■ MSG_GO_SET_LINE_COLOR

18.7

```
void     MSG_GO_SET_LINE_COLOR(
         byte   red,
         byte   green,
         byte   blue);
```

This message sets a GrObj's line color.

Source: Unrestricted.

Destination: GrObj.

Parameters:	*red*	The new color's red component.
	green	The new color's green component.
	blue	The new color's blue component.

Return: Nothing.

Interception: Unlikely.

■ MSG_GO_SET_LINE_MASK

```
void     MSG_GO_SET_LINE_MASK(
         SystemDrawMask mask);
```

This message sets a GrObj's line mask.

Source: Unrestricted.

Destination: GrObj.

Parameters:	*mask*	The new line draw mask.

Return: Nothing.

Interception: Unlikely.

■ MSG_GO_SET_LINE_END

```
void     MSG_GO_SET_LINE_END(
         LineEnd end);
```

This message sets a GrObj's line end.

Source: Unrestricted.

Destination: GrObj.

Parameters: *end* The new line end.

Return: Nothing.

Interception: Unlikely.

■ MSG_GO_SET_LINE_JOIN

```
void	MSG_GO_SET_LINE_JOIN(
	LineJoin join);
```

This message sets a GrObj's line join.

Source: Unrestricted.

Destination: GrObj.

Parameters: *join* The new line join.

Return: Nothing.

Interception: Unlikely.

■ MSG_GO_SET_LINE_STYLE

```
void	MSG_GO_SET_LINE_DRAW_STYLE(
	MixMode mode);
```

This message sets a GrObj's line style, or "dottedness."

Source: Unrestricted.

Destination: GrObj.

Parameters: *style* The new line style.

Return: Nothing.

Interception: Unlikely.

■ MSG_GO_SET_LINE_WIDTH

```
void	MSG_GO_SET_LINE_WIDTH(
	WWFixed width);
```

This message sets a GrObj's line width.

Source: Unrestricted.

Destination: GrObj.

Parameters: *width* The new line width.

Return: Nothing.

Interception: Unlikely.

■ MSG_GO_SET_LINE_MITER_LIMIIT

```
void      MSG_GO_SET_LINE_MITER_LIMIT(
          WWFixed miterLimit);
```

This message sets a GrObj's line miter limit, used with mitered line joins.

Source: Unrestricted.

Destination: GrObj.

Parameters: *miterLimit* The new miter limit.

Return: Nothing.

Interception: Unlikely.

■ MSG_GO_INIT_TO_DEFAULT_ATTRS

```
void      MSG_GO_INIT_TO_DEFAULT_ATTRS ();
```

This message requests that the object initialize its attributes to the current defaults.

Source: Unrestricted.

Destination: GrObj.

Parameters: None.

Return: Nothing.

Interception: Unlikely.

18.7.2.4 Action Notification

The following messages allow the application to specify what sort, if any, of notification should be sent out when the given GrObj is changed.

■ MSG_GO_SET_ACTION_NOTIFICATION_OUTPUT

```
void      MSG_GO_SET_ACTION_NOTIFICATION_OUTPUT(
          optr                object,
          Message             messageNumber);
```

This message specifies the message and OD for the GrObj to send notification to when an action is performed on it.

Source: Unrestricted.

Destination: GrObjBody.

Parameters: *object* The object which should be notified. NULL to clear the output.

messageNumber The message to send.

Return: Nothing.

Interception: Unlikely.

■ MSG_GO_SUSPEND_ACTION_NOTIFICATION

```
void    MSG_GO_SUSPEND_ACTION_NOTIFICATION();
```

This message suspends action notification for a GrObj. This prevents the GrObj from sending out any action notification.

Source: Unrestricted.

Destination: GrObj.

Parameters: None.

Return: Nothing.

Interception: Unlikely.

■ MSG_GO_UNSUSPEND_ACTION_NOTIFICATION

```
void    MSG_GO_UNSUSPEND_ACTION_NOTIFICATION();
```

This message counteracts MSG_GO_SUSPEND_ACTION_NOTIFICATION. If all suspends have been balanced, the GrObj will be free to send out action notification. However, action notifications that were aborted during the suspend period will not be sent out.

Source: Unrestricted.

Destination: GrObj.

Parameters: None.

Return: Nothing.

Interception: Unlikely.

■ MSG_GO_NOTIFY_ACTION

```
word      MSG_GO_NOTIFY_ACTION (
          GrObjActionNotificationType *action);
```

This message is sent to an object after it has been added to a GrObjBody.

Source: Generally the object itself.

Destination: GrObj.

Parameters: *action* What sort of operation was performed.

Return: Word of data whose meaning depends on the notification type.

Interception: Possible. Call superclass before doing own processing.

18.7.2.5 Transformations

■ MSG_GO_FLIP_HORIZ

```
void      MSG_GO_FLIP_HORIZ ();
```

This message flips the GrObj about its vertical axis.

Source: Unrestricted.

Destination: GrObj.

Parameters: None.

Return: Nothing.

Interception: Unlikely.

■ MSG_GO_FLIP_VERT

```
void      MSG_GO_FLIP_VERT ();
```

This message flips the GrObj about its horizontal axis.

Source: Unrestricted.

Destination: GrObj.

Parameters: None.

Return: Nothing.

Interception: Unlikely.

■ MSG_GO_ROTATE

```
void      MSG_GO_ROTATE (
          WWFixed                      angle,
          GrObjHandleSpecification     center);
```

This message rotates the GrObj about one of its handles.

Source: Unrestricted.

Destination: GrObj.

Parameters: *angle* Angle of rotation, in degrees counterclockwise.

center Which of the GrObj's handles is the center of rotation.

Return: Nothing.

Interception: Unlikely.

■ MSG_GO_MOVE

```
void      MSG_GO_MOVE (
          PointDWFixed *distance);
```

This message moves a GrObj to a relative position.

Source: Unrestricted.

Destination: GrObj.

Parameters: *distance* Offsets by which to displace the object.

Return: Nothing.

Interception: Unlikely.

■ MSG_GO_MOVE_CENTER_ABS

```
void      MSG_GO_MOVE_CENTER_ABS (
          PointDWFixed *location);
```

This message moves a GrObj to an absolute position.

Source: Unrestricted.

Destination: GrObj.

Parameters: *location* Object's new location.

Return: Nothing.

Interception: Unlikely.

■ MSG_GO_NUDGE

```
void      MSG_GO_NUDGE (
          sword  xDistance.
          sword  yDistance);
```

Move the GrObj by a number of device units.

Source: Unrestricted.

Destination: GrObj.

Parameters:	*xDistance*	Horizontal offset.
	yDistance	Vertical offset

Return: Nothing.

Interception: Unlikely.

■ MSG_GO_SET_SIZE

```
void      MSG_GO_SET_SIZE (
          PointWWFixed *size);
```

Set the object's width and height in points. The dimensions are calculated by mapping the object's corners into document coordinates and calculating the distances between them. The center of the selection handles of a rectangle represent the corners mapped into document coordinates. The line thickness is not included in this calculation.

Source: GrObjBody only.

Destination: GrObj.

Parameters:	*size*	The GrObj's new size.

Return: Nothing.

Interception: Possible. Call superclass before doing own processing.

■ MSG_GO_SET_POSITION

```
void      MSG_GO_SET_POSITION (
          PointDWFixed *location);
```

Set the position of the upper left of a GrObj. The position set is in document coordinates unless the GrObj is in a group, in which case the position is in coordinates relative to the group's upper left corner. If the GrObj has been rotated, skewed, or otherwise transformed, this sets the location of the selection handle that was originally at the upper left of the GrObj.

Source: GrObjBody only.

Destination: GrObj.

Parameters: None.

Return: Nothing.

Interception: Possible. Call superclass before doing own processing.

■ MSG_GO_SCALE

18.7

```
void        MSG_GO_SCALE(
            GrObjAnchoredScaleData *params);
```

This message scales a GrObj.

Source: Unrestricted.

Destination: GrObj.

Parameters: *params* A structure containing scale factors and which handle will act as the center of scaling.

Return: Nothing.

Structures:

```
typedef struct {
      WWFixed GOSD_xScale;
      WWFixed GOSD_yScale;
} GrObjScaleData;
typedef struct {
      GrObjScaleData GOASD_scale;
      GrObjHandleSpecification GOASD_scaleAnchor;
} GrObjAnchoredScaleData;
```

Interception: Possible. Call superclass before doing own processing.

■ MSG_GO_SKEW

```
void        MSG_GO_SKEW(
            GrObjAnchoredSkeweData *params);
```

This message skews a GrObj.

Source: Unrestricted.

Destination: GrObj.

Parameters: *params* A structure containing skewage amounts and which handle will act as the center of scaling.

Return: Nothing.

Structures:

```
typedef struct {
      WWFixed GOSD_xDegrees; /* counter-clockwise */
      WWFixed GOSD_yDegrees;
} GrObjSkewData;
```

```
typedef struct {
      GrObjSkewData GOASD_degrees;
      GrObjHandleSpecification GOASD_skewAnchor;
} GrObjAnchoredSkewData;
```

Interception: Possible. Call superclass before doing own processing.

■ MSG_GO_GET_SIZE

```
void      MSG_GO_GET_SIZE (
          GOGetSizeParams *retVal);
```

Get the object's width and height in points. The dimensions are calculated by mapping the object's corners into document coordinates and calculating the distances between them. The center of the selection handles of a rectangle represent the corners mapped into document coordinates. The line thickness is not included in this calculation.

Source: GrObjBody only.

Destination: GrObj.

Parameters: *retVal* Structure to hold return value.

Return: Nothing explicitly.

retVal Structure filled with size information.

Interception: Possible. Call superclass before doing own processing.

Structures:

```
typedef struct {
       WWFixed GOGSP_height;
       WWFixed GOGSP_width;
} GOGetSizeParams;
```

◆ Objects

■ MSG_GO_GET_POSITION

```
void      MSG_GO_GET_POSITION (
          PointDWFixed *retValue);
```

Get the position of the upper left of a GrObj. The position is in document coordinates unless the GrObj is in a group, in which case the position is relative to the upper left of the group. If the GrObj has been rotated, skewed, or otherwise transformed, then this message gets the location of the selection handle that was originally at the upper left of the GrObj.

Source: GrObjBody only.

Destination: GrObj.

Parameters: *retValue* Structure to hold return value.

Return: Nothing explicitly.

retValue Structure filled with location information.

Interception: Possible. Call superclass before doing own processing.

18.7.2.6 Cutting, Pasting, and Transfer Items

The following messages deal with the structures used with GrObj transfer items. You will also use the **GrObjTransferParams** structure:

```
typedef struct {
      StyleSheetParams  GTP_ssp;
      VisTextSaveStyleSheetParams GTP_textSSP;
      PointDWFixed      GTP_selectionCenterDOCUMENT;
      Handle            GTP_optBlock;
      Handle            GTP_vmFile;
      word              GTP_curSlot;
      dword             GTP_id;
      word              GTP_curSize;
      word              GTP_curPos;
} GrObjTransferParams;
```

■ MSG_GO_CREATE_TRANSFER

```
void MSG_GO_CREATE_TRANSFER(
        GrObjTransferParams _far *params);
```

Any subclass that requires more than just instance data to reconstruct itself will subclass this message to construct the **VMChain** necessary to do so.

The *GTP_curSlot* field of params is updated to the next slot in the tree, specified by *GTP_treeBlock*.

■ MSG_GO_REPLACE_WITH_TRANSFER

```
void MSG_GO_REPLACE_WITH_TRANSFER(
        GrObjTransferParams _far *params);
```

This message causes an existing GrObj to read the passed transfer item and recreate itself from that information. It is sent to an object during paste-type operations.

Source: Unrestricted.

Destination: Any GrObj.

Interception: Any subclass that requires more than just instance data to reconstruct itself will subclass this message to parse the **VMChain** necessary to do so.

■ MSG_GO_WRITE_INSTANCE_TO_TRANSFER

```
void MSG_GO_WRITE_INSTANCE_TO_TRANSFER(
        GrObjTransferParams _far *params);
```

This message causes the GrObj to write any data needed to create the GrObj into a transfer item.

The *GTP_curSlost* field of params is updated to point just past the last written data so that the superclass can begin writing, etc.

Interception: Any subclass with data necessary to recreate the object will subclass this message. The subclass will first call its superclass, then write in its extra data.

■ MSG_GO_READ_INSTANCE_FROM_TRANSFER

```
void MSG_GO_READ_INSTANCE_FROM_TRANSFER(
          GrObjTransferParams _far *params);
```

This message causes an existing GrObj to read the passed transfer item and recreate itself from that information. It is sent to an object during paste-type operations.

Source: Unrestricted.

Destination: Any GrObj.

Interception: Any subclass with data necessary to reconstruct itself will subclass this message. The subclass will first call its superclass, then read in its extra data.

18.7.2.7 Miscellaneous Messages

■ MSG_GO_ADD_POTENTIAL_SIZE_TO_BLOCK

■ MSG_GO_SUBTRACT_POTENTIAL_SIZEFROM_BLOCK

■ MSG_GO_GET_GROBJ_CLASS

■ MSG_GO_BECOME_SELECTED

```
void      MSG_GO_BECOME_SELECTED(
          HandleUpdateMode hum);
```

■ MSG_GO_TOGGLE_SELECTION

```
void      MSG_GO_TOGGLE_SELECTION();
```

■ MSG_GO_BECOME_UNSELECTED

```
void      MSG_GO_BECOME_UNSELECTED();
```

■ MSG_GO_UNDRAW_SPRITE

■ MSG_GO_DRAW_SPRITE

■ **MSG_GO_DRAW_SPRITE_RAW**

■ **MSG_GO_DRAW_HANDLES**

■ **MSG_GO_UNDRAW_HANDLES**

■ **MSG_GO_DRAW_HANDLES_RAW**

■ **MSG_GO_DRAW_HANDLES_FORCE**

■ **MSG_GO_DRAW_HANDLES_MATCH**

■ **MSG_GO_DRAW_HANDLES_OPPOSITE**

■ **MSG_GO_ACTIVATE_MOVE**

■ **MSG_GO_ACTIVATE_RESIZE**

■ **MSG_GO_ACTIVATE_ROTATE**

■ **MSG_GO_ACTIVATE_CREATE**

■ **MSG_GO_REACTIVATE_CREATE**

■ **MSG_GO_START_CHOOSE_ABS**

■ **MSG_GO_START_MOVE_ABS**

■ **MSG_GO_JUMP_START_MOVE**

■ **MSG_GO_JUMP_START_RESIZE**

■ **MSG_GO_JUMP_START_ROTATE**

■ **MSG_GO_PTR_CHOOSE_ABS**

■ **MSG_GO_PTR_MOVE**

■ **MSG_GO_PTR_RESIZE**

■ **MSG_GO_PTR_ROTATE**

■ **MSG_GO_PTR_MOVE_ABS**

◆Objects

■ **MSG_GO_END_CHOOSE_ABS**

■ **MSG_GO_END_MOVE_ABS**

■ **MSG_GO_END_MOVE**

■ **MSG_GO_END_RESIZE**

■ **MSG_GO_END_ROTATE**

■ **MSG_GO_CLEAR**

■ **MSG_GO_INVERT_HANDLES**

■ **MSG_GO_INIT_BASIC_DATA**

■ **MSG_GO_ALIGN**

■ **MSG_GO_ALIGN_TO_GRID**

■ **MSG_GO_SEND_ANOTHER_TOOL_ACTIVATED**

■ **MSG_GO_ANOTHER_TOOL_ACTIVATED**

■ **MSG_GO_SPECIAL_RESIZE_CONSTRAIN**

```
void      MSG_GO_SPECIAL_RESIZE_CONSTRAIN(
          GrObjHandleSpecification grObjHandleSpec);
```

■ **MSG_GO_DUPLICATE_FLOATER**

■ **MSG_GO_GRAB_MOUSE**

■ **MSG_GO_RELEASE_MOUSE**

■ **MSG_GO_UNGROUPABLE**

■ **MSG_GO_GET_BOUNDING_RECTDWFIXED**

■ **MSG_GO_CALC_DOCUMENT_DIMENSIONS**

■ **MSG_GO_INIT_CREATE**

■ **MSG_GO_INVERT_GROBJ_SPRITE**

■ MSG_GO_INVERT_GROBJ_NORMAL_SPRITE

■ MSG_GO_INVALIDATE

■ MSG_GO_GET_DW_PARENT_BOUNDS

■ MSG_GO_GET_DWF_PARENT_BOUNDS

18.7 ■ MSG_GO_GET_WWF_PARENT_BOUNDS

■ MSG_GO_GET_WWF_OBJECT_BOUNDS

```
void      MSG_GO_GET_WWF_OBJECT_BOUNDS(
          RectWWFixed         *retValue);
```

■ MSG_GO_BECOME_UNEDITABLE

■ MSG_GO_EVALUATE_POSITION

■ MSG_GO_EVALUATE_PARENT_POINT

■ MSG_GO_GET_CENTER

```
void      MSG_GO_GET_CENTER(
          PointDWFixed        *center);
```

■ MSG_GO_DRAW

■ MSG_GO_LARGE_START_SELECT

■ MSG_GO_LARGE_START_MOVE_COPY

■ MSG_GO_LARGE_END_SELECT

■ MSG_GO_LARGE_END_MOVE_COPY

■ MSG_GO_LARGE_DRAG_SELECT

■ MSG_GO_LARGE_DRAG_MOVE_COPY

■ MSG_GO_LARGE_PTR

■ MSG_GO_AFTER_ADDED_TO_GROUP

■ MSG_GO_BEFORE_REMOVED_FROM_GROUP

◆ Objects

■ MSG_GO_SUBST_AREA_TOKEN

■ MSG_GO_SUBST_LINE_TOKEN

■ MSG_GO_GET_ANCHOR_DOCUMENT

■ MSG_GO_BECOME_EDITABLE

■ MSG_GO_DRAW_EDIT_INDICATOR

■ MSG_GO_UNDRAW_EDIT_INDICATOR

■ MSG_GO_DRAW_EDIT_INDICATOR_RAW

■ MSG_GO_INVERT_EDIT_INDICATOR

■ MSG_GO_GROBJ_SPECIFIC_INITIALIZE

■ MSG_GO_GROBJ_SPECIFIC_INITIALIZE_WITH_DATA_BLOCK

■ MSG_GO_APPLY_ATTRIBUTES_TO_GSTATE

■ MSG_GO_COMBINE_AREA_NOTIFICATION_DATA

```
@message void MSG_GO_COMBINE_AREA_NOTIFICATION_DATA(
          Handle /* GrObjNotifyAreaAttrChange */ change);
```

Combine this GrObj's attributes with the passed structure.

Structures:

```
typedef struct {
      GrObjBaseAreaAttrElement GNAAC_areaAttr;
      GrObjBaseAreaAttrDiffs GNAAC_areaAttrDiffs;
} GrObjNotifyAreaAttrChange;

typedef WordFlags GrObjBaseAreaAttrDiffs;
#define GOBAAD_MULTIPLE_ELEMENT_TYPES 0x8000
#define GOBAAD_MULTIPLE_STYLE_ELEMENTS 0x4000
#define GOBAAD_MULTIPLE_COLORS 0x2000
#define GOBAAD_MULTIPLE_BACKGROUND_COLORS 0x1000
#define GOBAAD_MULTIPLE_MASKS 0x0800
#define GOBAAD_MULTIPLE_PATTERNS 0x0400
#define GOBAAD_MULTIPLE_DRAW_MODES 0x0200
```

```
#define GOBAAD_MULTIPLE_INFOS 0x0100
#define GOBAAD_MULTIPLE_GRADIENT_END_COLORS 0x0080
#define GOBAAD_MULTIPLE_GRADIENT_TYPES 0x0040
#define GOBAAD_MULTIPLE_GRADIENT_INTERVALS 0x0020
#define GOBAAD_FIRST_RECIPIENT 0x0001
      /* A grobj knows that it's the first one to
       * receive this data buffer if this flag is
       * set (and should clear it). */
```

18.7

■ MSG_GO_COMBINE_LINE_NOTIFICATION_DATA

■ MSG_GO_COMBINE_SELECT_STATE_NOTIFICATION_DATA

■ MSG_GO_COMBINE_SELECTION_STATE_NOTIFICATION_DATA

■ MSG_GO_COMBINE_STYLE_NOTIFICATION_DATA

```
void MSG_GO_COMBINE_STYLE_NOTIFICATION_DATA(
        Handle change);     /* handle of block containing NotifyStyleChange */
```

■ MSG_GO_COMBINE_STYLE_SHEET_NOTIFICATION_DATA

```
void MSG_GO_COMBINE_STYLE_SHEET_NOTIFICATION_DATA(
         Handle change );   /* handle of NotifyStyleSheetChange */
```

■ MSG_GO_SEND_UI_NOTIFICATION

■ MSG_GO_COMPLETE_TRANSFORM

■ MSG_GO_COMPLETE_TRANSLATE

■ MSG_GO_INSERT_OR_DELETE_SPACE

■ MSG_GO_EVALUATE_PARENT_POINT_FOR_EDIT

■ MSG_GO_GET_POINTER_IMAGE

18.7.3 Shape Classes

The GrObj class on its own, while very powerful, is rather general. GrObjs per se aren't actually associated with any shape. Thus, a number of subclasses exist to correctly manifest as different kinds of shapes. None of these classes has any class-specific messages, although each of course must subclass many messages to correctly maintain instance data and draw correctly.

RectClass is a subclass of **GrObjClass** with no specialized instance data or messages.

RoundedRectClass is a subclass of **RectClass** with one piece of additional instance data: *RRI_radius*.

18.7

LineClass is a subclass of **GrObjClass** with no specialized instance data or messages.

GStringClass is a subclass of **GrObjClass** which does not correspond to any shape. Instead, it may hold any GString. It has two extra instance fields: *GSI_vmemBlockHandle* is the VMBlockHandle in which the GString is stored. The *GSI_gstringCenterTrans* field holds the GString's center point. It has one extra message, MSG_GSO_SET_GSTRING, which sets the contents of the GString.

18.7.4 GroupClass

The **GroupClass** is a subclass of **GrObjClass** which is used to group GrObjs. Grouped objects may be moved, rotated, scaled, and skewed collectively.

Code Display 18-24 GroupClass Instance Data

```
@instance CompPart              GI_drawHead;
@instance word                  GI_suspendCount;
@instance GroupUnsuspendOps     GI_unsuspendOps;

typedef ByteFlags GroupUnsuspendOps;
#define GUO_EXPAND 0x01
```

■ MSG_GROUP_ADD_GROBJ

■ MSG_GROUP_REMOVE_GROBJ

■ MSG_GROUP_CREATE_GSTATE

■ MSG_GROUP_PROCESS_ALL_GROBJS_SEND_CALL_BACK_MESSAGE

18.7

■ MSG_GROUP_INITIALIZE

■ MSG_GROUP_EXPAND

■ MSG_GROUP_INSTANTIATE_GROBJ

```
optr      MSG_GROUP_INSTANTIATE_GROBJ(
          ClassStruct        *class);
```

18.7.5 PointerClass

PointerClass is a subclass of **GrObjClass**. GeoDraw's Arrow Pointer is a member of this class. Pointers are used for selecting graphic objects and interacting with their handles. Pointers control moves and resizes.

RotatePointerClass is a subclass of **PointerClass**. Like pointer class, it works with an object's handles, but it rotates instead of resizing. The **BitmapPointerClass** is useful for working with bitmaps in a GrObj context.

Code Display 18-25 PointerClass Instance Data

```
@instance PointerModes                    PTR_modes; /* Internal */
```

18.7.6 GrObjVisGuardian Classes

GrObj provides the ability to incorporate vis objects into a graphic layer. This requires that two special classes be set up, known as a guardian and a ward. The GrObjVisGuardian is a subclass of GrObj, and accepts all standard

GrObj messages. The ward class is a subclass of **GrObjVisClass**, set up to work with the appropriate visual class. **GrObjVisClass** is a master variant class and uses the appropriate visual class as its superclass.

Thus, there is a **SplineGuardianClass** and a **GrObjSplineClass** for working with splines and polylines. **TextGuardianClass** and **GrObjTextClass** work with text objects. Bitmaps are implemented using **BitmapGuardianClass** together with the **GrObjBitmap** class.

Working with these object pairs is simple: you can work with the guardian object as you would with any graphic object. This happens automatically and transparently. You may also query any guardian object for the OD of its ward, and then interact with the ward as with a normal object of the appropriate visual type. Thus, you could send a MSG_GOVG_GET_VIS_WARD to a TextGuardian, then pass a MSG_VIS_TEXT_REPLACE_ALL to the returned OD.

The VisText object is documented in “The Text Objects,” Chapter 10.

Chances are, you will never be concerned with the inner workings of the **GrObjVisClass**. It is a subclass of VisClass, with one extra piece of instance data: *GVI_guardian*, which holds the optr of the ward’s guardian object.

Code Display 18-26 GrObjVisGuardianClass Instance Data

```
@instance optr                          GOVGI_ward;
@instance word                          *GOVGI_class;
@instance GrObjVisGuardianFlags         GOVGI_flags;
```

■ MSG_GOVG_UPDATE_VIS_WARD_WITH_STORED_DATA

■ MSG_GOVG_CONVERT_LARGE_MOUSE_DATA

■ MSG_GOVG_CREATE_VIS_WARD

```
optr      MSG_GOVG_CREATE_VIS_WARD(
          MemHandle          wardBlock);
```

■ MSG_GOVG_ADD_VIS_WARD

■ MSG_GOVG_APPLY_OBJECT_TO_VIS_TRANSFORM

■ MSG_GOVG_CREATE_GSTATE

■ MSG_GOVG_VIS_BOUNDS_SETUP

■ MSG_GOVG_SET_VIS_WARD_MOUSE_EVENT_TYPE

■ MSG_GOVG_UPDATE_EDIT_GRAB_WITH_STORED_DATA

■ MSG_GOVG_NORMALIZE

■ MSG_GOVG_NOTIFY_VIS_WARD_CHANGE_BOUNDS

■ MSG_GOVG_GET_EDIT_CLASS

■ MSG_GOVG_GET_VIS_WARD_OD

```
optr      MSG_GOVG_GET_VIS_WARD_OD();
```

■ MSG_GOVG_GET_TRANSFER_BLOCK_FROM_VIS_WARD

■ MSG_GOVG_CREATE_WARD_WITH_TRANSFER

■ MSG_GOVG_SET_VIS_WARD_CLASS

GrObjVisClass Messages

■ MSG_GV_GET_WWFIXED_CENTER

■ MSG_GV_SET_GUARDIAN_LINK

■ MSG_GV_SET_VIS_BOUNDS

Ruler Object Library

19

The ruler library provides objects that act as rulers. These rulers may be linked up with a GenView object so that when the user scrolls around within the GenView, the rulers update accordingly. The rulers work with a variety of measurement systems and also provide grids and guidelines to help the user to figure out the spacial layout of a graphic space. To aid the user's mouse movement, the rulers can constrain the mouse's movement to certain lines.

To work with the basic ruler set-up, you don't need to know much about GEOS other than the basics of objects and messaging. However, for certain customizations, you will probably need to know something about GenViews and notifications they pass on to linked objects.

19.1 Ruler Features

Rulers are a "must have" feature for drawing programs or any geode concerned with layout-intensive tasks. By alerting the rulers to pointer events, the application may display marks in the ruler which track the mouse's position so that the user knows the exact mouse coordinates. The ruler library provides many more features to applications prepared to support those features:

◆ Grids
The ruler's tick marks are of course drawn inside the ruler. However, the ruler can draw a grid over the main view so that the user may see where major units fall on the document directly.

◆ Guides
While a regular grid is helpful, the user may want to set up guidelines to mark off areas important to a document's layout. These guides may appear both as marks on the ruler and lines across the main view itself.

◆ Mouse Constraints
The rulers can act as mouse constraint managers. The mouse location can snap to grid lines, guidelines, or lines at any angle through any point.

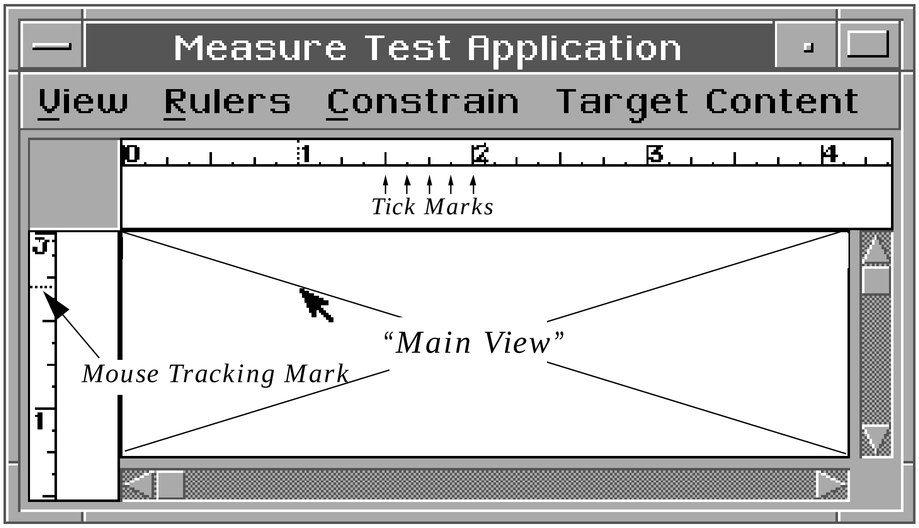


Figure 19-1 *Rulers*

- Large Document Support
  The rulers work with large (32-bit) documents. They respond correctly to large mouse and scrolling events. However, at this time there is no way using C to intercept the message by which the Ruler handles large pointer events.

- Controller Support
  The ruler library includes a full suite of controller objects, including controllers for managing grids and guides.

19.2 Ruler Setup

For each ruler a geode displays, it will need to include three objects: an object of **RulerViewClass** to display the ruler, a **RulerContentClass** object to act as the top of the RulerView's visible tree, and a **VisRulerClass** object, the ruler itself. Also, the geode may wish to include some of the available ruler control objects.

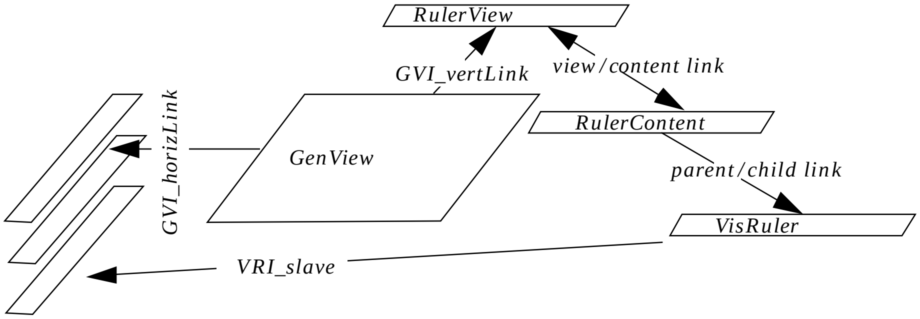


Figure 19-2 *UI Linkages for Rulers*

The RulerViews should be alerted when the main GenView scrolls or scales, so set the GenView's *GVI_horizLink* and *GVI_vertLinks* to hold the optrs of the RulerViews. A RulerView to the side of the GenView should be linked by the horizontal link; a RulerView above or below the GenView should be linked vertically.

Each RulerView should have a RulerContent as its content. Set this up as you would a normal GenView/VisContent linkage. The RulerContent functions as a VisContent for most purposes; there is some subclassed behavior so that the VisRuler will be notified when the RulerView has detected a scroll or scale event.

A VisRuler should be the child of the RulerContent. If you have more than one VisRuler associated with a view, you should use the *VRI_slave* links to connect them. When one VisRuler receives certain messages, it will handle them and pass them on to whatever VisRuler has been designated in its *VRI_slave* field. If this slave ruler itself has a slave, then the message will be relayed again, and so on. Thus the application only has to send messages to one ruler, instead of to all of them. The ruler which is not the target of a *VRI_slave* link is the view's master ruler, and its VRA_MASTER bit should be set.

If the view only has one ruler associated with it, that ruler's VRA_MASTER bit should be set.

●

You cannot relay mouse events to a VisRuler across threads.

If you will use the ruler library's mouse constraint management support, then the VisRulers must be run in the same thread which manages the main content. The VisRuler messages which deal with mouse positions receive the mouse location parameters on the stack, and thus the handlers must be running in the same thread as the callers. This is not a major restriction: since mouse events are so common, sending them across threads to be processed would lead to slow, jerky responses.

Your application may also include other objects:

- RulerTypeControl to allow the user to use different measurement units.
- RulerGridControl to draw a grid on the main view to which the user may snap the mouse.
- RulerGuideControl to allow the user to manage guidelines.
- RulerShowControl to show or hide rulers. Because hidden rulers are set not usable, and thus do not work well with GCN, applications including a RulerShowControl need to do some extra work. This is not difficult, but exactly what needs to be done may vary between applications. For details, see Section 19.4.1 on page ● 1190.
- If you will be using any of the ruler controllers, you must provide a way to relay classed events to the VisRuler objects. Whichever object will have the target when the controllers send out their classed events must have a special handler for MSG_META_SEND_CLASSED_EVENT which determines if the message is intended for a member of VisRuler class, and if so passes it to the master VisRuler object. Note that if you are working with the grobj library, a targeted GrObjBody correctly relays classed events to rulers.

Finally, if you wish the VisRuler to take special action on certain mouse events (perhaps providing a mark on the ruler to track the mouse pointer's position), the GenView's content should intercept those mouse events and send messages to the VisRuler (again, the VisRuler at the top of the *VRI_slave* chain) with the requested action.

19.3 VisRuler Instance Data

The VisRuler's instance data fields are listed below. Depending on your setup, the management of most of these fields will probably be taken care of by the ruler itself or some of its associated controllers. However, many programs set values for the *VRI_rulerAttrs*, *VRI_constrainStrategy*, and *VRI_slave* fields when defining their rulers.

Code Display 19-1

```
@classVisRulerClass,VisClass;
@instance VisRulerAttributes VRI_rulerAttrs;

typedef ByteFlags VisRulerAttributes;
#define VRA_IGNORE_ORIGIN        0x80
#define VRA_SHOW_GUIDES          0x40
#define VRA_SHOW_GRID            0x20
#define VRA_SHOW_MOUSE           0x10
#define VRA_HORIZONTAL           0x08
#define VRA_MASTER               0x04

@instance VisRulerType VRI_type;

typedef ByteEnum VisRulerType;
#define VRT_INCHES      0x0
#define VRT_CENTIMETERS 0x1
#define VRT_POINTS      0x2
#define VRT_PICAS       0x3
#define VRT_CUSTOM      CUSTOM_RULER_DEFINITION
#define VRT_NONE        NO_RULERS
#define VRT_DEFAULT     SYSTEM_DEFAULT

@instance VisRulerConstrainStrategy VRI_constrainStrategy;

typedef WordFlags VisRulerConstrainStrategy;
#define VRCS_OVERRIDE                                    0x8000
#define VRCS_SET_REFERENCE                               0x2000
#define VRCS_SNAP_TO_GRID_X_ABSOLUTE                     0x1000
#define VRCS_SNAP_TO_GRID_Y_ABSOLUTE                     0x0800
#define VRCS_SNAP_TO_GRID_X_RELATIVE                     0x0400
#define VRCS_SNAP_TO_GRID_Y_RELATIVE                     0x0200
#define VRCS_SNAP_TO_GUIDES_X                            0x0100
#define VRCS_SNAP_TO_GUIDES_Y                            0x0080
#define VRCS_CONSTRAIN_TO_HORIZONTAL_AXIS                0x0040
#define VRCS_CONSTRAIN_TO_VERTICAL_AXIS                  0x0020
#define VRCS_CONSTRAIN_TO_UNITY_SLOPE_AXIS               0x0010
#define VRCS_CONSTRAIN_TO_NEGATIVE_UNITY_SLOPE_AXIS      0x0008
#define VRCS_CONSTRAIN_TO_VECTOR                         0x0004
#define VRCS_CONSTRAIN_TO_VECTOR_REFLECTION              0x0002
#define VRCS_INTERNAL                                    0x0001

@instance MinIncrementType VRI_minIncrement; /* minimum increment displayed */
```

```
typedef union {
        MinUSMeasure            MIT_US;
        MinMetricMeasure        MIT_METRIC;
        MinPointMeasure         MIT_POINT;
        MinPicaMeasure          MIT_PICA;
} MinIncrementType;

typedef ByteEnum MinUSMeasure;
#define MUSM_EIGHTH_INCH        0x0
#define MUSM_QUARTER_INCH       0x1
#define MUSM_HALF_INCH          0x2
#define MUSM_ONE_INCH           0x3

typedef ByteEnum MinMetricMeasure;
#define MMM_MILLIMETER          0x0
#define MMM_HALF_CENTIMETER     0x1
#define MMM_CENTIMETER          0x2

typedef ByteEnum MinPointMeasure;
#define MPM_25_POINT            0x0
#define MPM_50_POINT            0x1
#define MPM_100_POINT           0x2

typedef ByteEnum MinPicaMeasure;
#define MPM_PICA                0x0
#define MPM_INCH                0x1

@instance WWFixed VRI_scale; /* scale factor */

@instance DWFixed VRI_origin; /* 0,0 of the ruler in document coordinates */

@instance PointDWFixed VRI_reference;

@instance sdword VRI_mouseMark;

@instance word VRI_window; /* actually a noreloc WindowHandle */

@instance optr VRI_slave;

@instance Grid VRI_grid;

@instance WWFixed VRI_vectorSlope; /* slope of vector */

@instance optr VRI_guideArray;

@instance word VRI_guideInfluence; /* influence of guides in pixels */

@instance word VRI_desiredSize;

@instance word VRI_reserved;

@instance dword VRI_invalOD; /* actually a noreloc optr */
```

◆Objects

```
@instance word VRI_transformGState; /* actually a noreloc GStateHandle */
```

The *VRI_rulerAttrs* field contains flags which determine whether the ruler will support given features. This field tends to vary between applications; put some thought into which flags to turn on.

VRA_IGNORE_ORIGIN
Using the *VRI_origin* field, it is possible to set an origin for the user to use based upon the coordinates of the graphic space within the main view. This can come in handy when the zero point for the ruler shouldn't correspond to the very left or top of the view; GeoWrite uses this when the user wants the ruler's origin to be based upon the right edge of a text object rather than the edge of the view. The VRA_IGNORE_ORIGIN flag effectively turns off this feature.

VRA_SHOW_GUIDES
This flag indicates that the user wants guidelines drawn over their main view.

VRA_SHOW_GRID
This flag indicates that the user wants gridlines drawn over the main view.

VRA_SHOW_MOUSE
This flag indicates that the ruler should draw a tick mark to show the mouse pointer's location.

VRA_HORIZONTAL
This flag indicates that the ruler is horizontal. Horizontal rulers should have this flag turned on; vertical rulers should not.

VRA_MASTER
If a given view has only one ruler associated with it, that ruler should have this flag set. If a given view has more than one ruler (perhaps having one horizontal and one vertical), one of these rulers should be set up as the master ruler. It should have the VRA_MASTER flag set and should have the second ruler's optr in its *VRI_slave* field. If there is a third ruler, the second ruler should have its optr in its (the second ruler's) VRI_slave field, and so on.

The *VRI_constrainStrategy* field contains the default mouse constraint strategy the ruler should use when constraining mouse movement.

VRCS_OVERRIDE
This flag isn't really meant to be part of the default mouse constraint; it will be ignored if in the *VRI_constrainStrategy* field. During some events other constrain strategies may be combined with the default mouse constrain strategy; if the other strategy's VRCS_OVERRIDE flag is set, then the default mouse constraint strategy field will be ignored.

VRCS_SET_REFERENCE
This flag signals that the next selection should signal that the user wishes the point clicked on to be the reference point, which is used with certain kinds of mouse constraint.

VRCS_SNAP_TO_GRID_X_ABSOLUTE
This flag signals that the mouse position should be snapped to the grid horizontally.

VRCS_SNAP_TO_GRID_Y_ABSOLUTE
This flag signals that the mouse position should be snapped to the grid vertically.

VRCS_SNAP_TO_GRID_X_RELATIVE
This flag signals that the mouse position should be horizontally snapped to a grid such as would be formed if the present grid spacing was used with the reference point as the grid's origin.

VRCS_SNAP_TO_GRID_Y_RELATIVE
This flag signals that the mouse position should be vertically snapped to a grid such as would be formed if the present grid spacing was used with the reference point as the grid's origin.

VRCS_SNAP_TO_GUIDES_X
This flag signals that the mouse positions should be horizontally snapped to guidelines. Note that guidelines are only effective within range of their "influence" in pixels.

VRCS_SNAP_TO_GUIDES_Y
This flag signals that the mouse positions should be vertically snapped to guidelines. Note that guidelines are only effective within range of their "influence" in pixels.

Objects

VRCS_CONSTRAIN_TO_HORIZONTAL_AXIS
This flag signals that the mouse position should be restrained to the horizontal line passing through the reference point.

VRCS_CONSTRAIN_TO_VERTICAL_AXIS
This flag signals that the mouse position should be restrained to the vertical line passing through the reference point.

VRCS_CONSTRAIN_TO_UNITY_SLOPE_AXIS
This flag signals that the mouse position should be restrained to the diagonal line with slope ($y = x$) passing through the reference point.

VRCS_CONSTRAIN_TO_NEGATIVE_UNITY_SLOPE_AXIS
This flag signals that the mouse position should be restrained to the diagonal line with slope ($y = -x$) passing through the reference point.

VRCS_CONSTRAIN_TO_VECTOR
This flag signals that the mouse position should be constrained to the line passing through the reference and vector points.

VRCS_CONSTRAIN_TO_VECTOR_REFLECTION
This flag signals that the mouse position should be constrained to the reflection of the line passing through the reference and vector points.

The *VRI_type* and *VRI_minIncrement* fields determine the measurement type and ruler increment of the ruler.Normally, the user will work with the ruler's type via a RulerTypeControl. **RulerTypeControlClass**'s features structures are shown below.

Code Display 19-2 RulerTypeControl Features

```
typedef WordFlags RTCFeatures;
#define RTCF_DEFAULT            (0x20)
#define RTCF_SPREADSHEET        (0x10)
#define RTCF_INCHES             (0x08)
#define RTCF_CENTIMETERS        (0x04)
#define RTCF_POINTS             (0x02)
#define RTCF_PICAS              (0x01)
#define RTC_DEFAULT_FEATURES \
        (RTCF_INCHES | RTCF_CENTIMETERS | RTCF_POINTS | RTCF_PICAS | RTCF_DEFAULT)
```

```
typedef WordFlags RTCToolboxFeatures;
#define RTCTF_DEFAULT           (0x20)
#define RTCTF_SPREADSHEET       (0x10)
#define RTCTF_INCHES            (0x08)
#define RTCTF_CENTIMETERS       (0x04)
#define RTCTF_POINTS            (0x02)
#define RTCTF_PICAS             (0x01)
#define RTC_DEFAULT_TOOLBOX_FEATURES (RTCF_INCHES | RTCF_CENTIMETERS)
```

19.3

The RulerTypeControl uses a notification block with the following structure:

```
typedef struct RulerTypeNotificationBlock {
      VisRulerType RTNB_type;
} RulerTypeNotificationBlock;
```

The ruler must keep track of the main view's scale; if the main view has doubled in scale, the ruler should put its tick marks twice as far apart. This scale is stored in the *VRI_scale* field. The ruler automatically receives a MSG_VIS_RULER_VIEW_SCALE_FACTOR_CHANGED when the main view's scale changes. To set the scale independently of the main view scale, send a MSG_VIS_RULER_SET_SCALE.

The *VRI_origin* field allows you to set an origin for the ruler other than (0,0). The coordinates are given in terms of the main view's coordinate system.

The reference point, used for various types of mouse constraint, is stored in the *VRI_reference* field.

VRI_mouseMark contains the coordinate at which the mouse tick is currently being drawn.

The *VRI_window* field, initiated automatically, holds the window handle of the main view.

The *VRI_slave* field holds the ruler's slave ruler; see the documentation for VRA_MASTER, above, to learn about master and slave rulers.

The *VRI_grid* field contains the information used to maintain the grid associated with a ruler.

The *VRI_vectorSlope* defines the slope of a vector that may be used with vector mouse constraints.

Objects

The *VRI_guideArray* field keeps track of the location where guideline data is stored; this field is initiated automatically.

The *VRI_guideInfluence* contains the influence to use when constraining the mouse position to guidelines. Guidelines will only attract the mouse if it is within a certain distance of them—this distance is the “influence” of the guidelines. It is measured in pixels, not points; thus, the guide influence is independent of scale.

The *VRI_desiredSize* field contains the ruler’s idea of its ideal thickness. By default, this is set so that the ruler will have room to draw its numbers and tick marks normally; if you will be subclassing the VisRuler to add more gadgetry, you may wish to use a different value here.

The *VRI_invalOD* field contains the object descriptor of the object which should be redrawn when the ruler needs to draw something differently to the main view; by the ruler’s default behavior, this will be updated automatically via a MSG_VIS_RULER_GAINED_SELECTION.

The *VRI_transformGState* field contains the handle of a GState. This GState contains the transformation which is used to transform the coordinate system used with mouse constraints.

■ MSG_VIS_RULER_SET_TYPE

```
void      MSG_VIS_RULER_SET_TYPE(
          VisRulerType type);
```

This message sets the *VRI_type* field.

■ MSG_VIS_RULER_GET_TYPE

```
VisRulerType MSG_VIS_RULER_GET_TYPE();
```

This message returns the value in the *VRI_type* field.

■ MSG_VIS_RULER_SET_CONSTRAIN_STRATEGY

```
void MSG_VIS_RULER_SET_CONSTRAIN_STRATEGY(
          VisRulerConstrainStrategy  setflags,
          VisRulerConstrainStrategy  clearflags);
```

This message sets the default mouse constrain strategy for the ruler.

Parameters:		
	setflags	Flags to set.
	clearflags	Flags to clear.

Return: Nothing.

■ MSG_VIS_RULER_GET_CONSTRAIN_STRATEGY

```
VisRulerConstrainStrategy
                  MSG_VIS_RULER_GET_CONSTRAIN_STRATEGY();
```

This message gets the current default constrain strategy for the ruler.

■ MSG_VIS_RULER_SET_IGNORE_ORIGIN

19.3

```
void MSG_VIS_RULER_SET_IGNORE_ORIGIN(
          Boolean              ignore);
```

This message sets the “ignore origin” state.

Parameters: *ignore* Set *true* (i.e. non-zero) to ignore the origin. Set *false* (i.e. zero) not to ignore the origin.

Return: Nothing.

■ MSG_VIS_RULER_SET_ORIGIN

```
void MSG_VIS_RULER_SET_ORIGIN(
          sdword               pointInt,
          word                 pointFrac);
```

Use this message to set the ruler’s origin point.

■ MSG_VIS_RULER_GET_ORIGIN

```
void MSG_VIS_RULER_GET_ORIGIN(
          DWFixedReturn *retval);
```

Use this message to retrieve the ruler’s origin point.

Structures:

```
typedef struct {
      word        unused;
      word        DWFR_frac;
      sdword      DWFR_int;
} DWFixedReturn;
```

■ MSG_VIS_RULER_SET_REFERENCE

```
void MSG_VIS_RULER_SET_REFERENCE( @stack
          sdword            yInt,
          word              yFrac,
          sdword            xInt,
          word              xFrac);
```

Set the ruler's reference point.

■ MSG_VIS_RULER_SET_VECTOR

```
void MSG_VIS_RULER_SET_VECTOR( @stack
          sdword            yInt,
          word              yFrac,
          sdword            xInt,
          word              xFrac);
```

Set the point to use to define the "vector"—together with the reference point, this will form a slope which may be used for mouse constraints.

■ MSG_VIS_RULER_SET_MIN_INCREMENT

```
void      MSG_VIS_RULER_SET_MIN_INCREMENT(
          MinIncrementType min);
```

This message sets the *VRI_minIncrement* field to the passed value. It must be passed the proper **MinIncrementType** value.

■ MSG_VIS_RULER_SET_SCALE

```
void      MSG_VIS_RULER_SET_SCALE(
          WWFixedAsDWord scale);
```

This message sets the *VRI_scale* field.

19.4 Managing Rulers

Most ruler functions will happen automatically. However, certain pieces of functionality require that the geode provide certain message handlers. Some geodes may have use for subclassed ruler objects, and subclassing will require further extra code. This section provides information for those who would learn more about how to manage rulers.

19.4.1 RulerShowControl

The RulerShowControl allows the user to show or hide rulers. This control works by setting RulerViews not usable. However, objects which aren't usable don't work with GCN, and thus you will need to set up some other object to intercept the appropriate notifications. Don't try to include a RulerShowControl unless you know about GCN or are willing to learn.

When declaring your RulerShowControl, you may choose a GCN list which the control will work with. The default list is MANUFACTURER_ID_GEOWORKS: GAGCNLT_DISPLAY_OBJECTS_WITH_RULERS. However, you may change this to be any list you like. You must also specify a message which some object will respond to.

Some object which will be usable when the user has access to the RulerShowControl should be on the control's GCN list. This object must be prepared to handle the message in the *RSCI_message* field. This message should take a RulerShowControlAttributes structure and set the ruler views usable or not usable accordingly. See Code Display 19-4 for an example.

Code Display 19-3 RulerShowControl Instance Data and Features

```
typedef WordFlags RSCCFeatures;
#define RSCCF_SHOW_VERTICAL (0x04)
#define RSCCF_SHOW_HORIZONTAL (0x02)
#define RSCCF_SHOW_RULERS (0x01)

typedef WordFlags RSCCToolboxFeatures;
/* There are no toolbox features. */

#define RSCC_DEFAULT_FEATURES (RSCCF_SHOW_VERTICAL | RSCCF_SHOW_HORIZONTAL)
#define RSCC_DEFAULT_TOOLBOX_FEATURES (0)

@instance RulerShowControlAttributes     RSCI_attrs;
typedef WordFlags RulerShowControlAttributes;
#define RSCA_SHOW_VERTICAL       0x8000
#define RSCA_SHOW_HORIZONTAL     0x4000

@instance GCNListType                   RSCI_gcnList; /* object to notify */
```

```
@instance Message                       RSCI_message; /* message to notify with */
```

Code Display 19-4 Sample Handler for use with RulerShowControl

```
@method MyDisplayClass, MSG_MD_UPDATE_RULERS
        /* The RulerShowControl’s RSCI_message field should be
         * MSG_MD_UPDATE_RULERS. The message should have been declared
         * using the RULER_SHOW_CONTROL_NOTIFY prototype. */

  if (attrs & RSCA_SHOW_VERTICAL) {
        @call LeftView::MSG_GEN_SET_USABLE(VUM_NOW);
  }
  else {
        @call LeftView::MSG_GEN_SET_NOT_USABLE(VUM_NOW);
  }
  if (attrs & RSCA_SHOW_HORIZONTAL) {
        @call TopView::MSG_GEN_SET_USABLE(VUM_NOW);
  }
  else {
        @call TopView::MSG_GEN_SET_NOT_USABLE(VUM_NOW);
  }
}
```

19.4.2 Mouse Tracking

The ruler has the ability to show a tick mark which tracks the mouse’s movement. This can be a boon to users who wish to know the mouse’s precise location.

■ MSG_VIS_RULER_SHOW_MOUSE

```
void MSG_VIS_RULER_SHOW_MOUSE();
```

Sets a flag such that the ruler will draw the mouse tick on pointer events.

■ MSG_VIS_RULER_HIDE_MOUSE

```
void MSG_VIS_RULER_HIDE_MOUSE();
```

Sets a flag such that the ruler won’t draw the mouse tick on pointer events.

■ MSG_VIS_RULER_DRAW_MOUSE_TICK

```
void MSG_VIS_RULER_DRAW_MOUSE_TICK( @stack
          sdword              yInt,
          word                yFrac,
          sdword              xInt,
          word                xFrac);
```

This message indicates that the ruler should draw a line indicating the mouse position and informs the ruler of the mouse's new position.

19.4.3 Grid Spacing and Constraint

Applications which provide rulers often do so to allow the user to work with the mouse more accurately. Such applications might also provide a grid. A grid helps locate the mouse pointer's exact location on the document without the need to watch the ruler. The ruler can also constrain mouse movement to the grid.

To allow the user to change the grid spacing, include a RulerGridControl. The features structures for the RulerGridControl are shown below.

Code Display 19-5 RulerGridControlClass Features

```
typedef WordFlags RGCFeatures;
#define RGCF_GRID_SPACING       (0x04)
#define RGCF_SNAP_TO_GRID       (0x02)
#define RGCF_SHOW_GRID          (0x01)

#define RGC_DEFAULT_FEATURES \
                        (RGCF_GRID_SPACING | RGCF_SNAP_TO_GRID | RGCF_SHOW_GRID)

#define RGC_DEFAULT_TOOLBOX_FEATURES (0)
```

■ MSG_VIS_RULER_SHOW_GRID

```
void MSG_VIS_RULER_SHOW_GRID();
```

Use this message to request that the grid be drawn to the main view, setting the VRA_SHOW_GRID flag.

■ MSG_VIS_RULER_HIDE_GRID

```
void MSG_VIS_RULER_HIDE_GRID();
```

Use this message to request that the grid not be drawn to the main view, clearing the VRA_SHOW_GRID flag.

■ MSG_VIS_RULER_DRAW_GRID

```
void MSG_VIS_RULER_DRAW_GRID(
        GStateHandle gstate);
```

This message will draw the grid to the passed GState if the VRA_SHOW_GRID_FLAG is set.

■ MSG_VIS_RULER_TURN_GRID_SNAPPING_ON

```
void MSG_VIS_RULER_TURN_GRID_SNAPPING_ON();
```

Set the default **VisRulerConstrainStrategy** to support grid snapping, setting the VRCS_SNAP_TO_GRID_X_ABSOLUTE and VRCS_SNAP_TO_GRID_Y_ABSOLUTE flags.

■ MSG_VIS_RULER_TURN_GRID_SNAPPING_OFF

```
void MSG_VIS_RULER_TURN_GRID_SNAPPING_OFF();
```

Set the default **VisRulerConstrainStrategy** to not include grid snapping, turning off the VRCS_SNAP_TO_GRID_X_ABSOLUTE and VRCS_SNAP_TO_GRID_Y_ABSOLUTE flags.

■ MSG_VIS_RULER_SET_GRID_SPACING

```
void MSG_VIS_RULER_SET_GRID_SPACING(
        WWFixedAsDWord spacing);
```

Set the grid's horizontal and vertical spacing, working with the *VRI_grid* field.

■ MSG_VIS_RULER_SET_HORIZONTAL_GRID_SPACING

```
void MSG_VIS_RULER_SET_HORIZONTAL_GRID_SPACING(
        WWFixedAsDWord spacing);
```

Set the horizontal grid spacing, leaving the vertical alone.

■ MSG_VIS_RULER_SET_VERTICAL_GRID_SPACING

```
void MSG_VIS_RULER_SET_VERTICAL_GRID_SPACING(
         WWFixedAsDWord spacing);
```

Set the vertical grid spacing, leaving the horizontal alone

■ MSG_VIS_RULER_GET_GRID_SPACING

```
void MSG_VIS_RULER_GET_GRID_SPACING(
         GridSpacing* gridspace);
```

This message returns the current grid spacing.

Structures: This message works with the **GridSpacing** structure. Do not confuse this structure with the **Grid** structure.

```
typedef struct {
      WWFixed GS_y;
      WWFixed GS_x;
} GridSpacing;
```

■ MSG_VIS_RULER_GET_STRATEGIC_GRID_SPACING

```
void MSG_VIS_RULER_GET_STRATEGIC_GRID_SPACING(
         GridSpacing* gridspace);
```

This message returns a grid spacing that will look nice on the screen at the current scale factor.

Structures: This message works with the **GridSpacing** structure. Do not confuse this structure with the **Grid** structure.

```
typedef struct {
      WWFixed GS_y;
      WWFixed GS_x;
} GridSpacing;
```

■ MSG_VIS_RULER_SNAP_TO_GRID

```
void MSG_VIS_RULER_SNAP_TO_GRID(
         PointDWFixed _far* point);
```

This message snaps the passed coordinate to the grid.

Parameters: *point* Point to snap to grid.

Return: Nothing returned explicitly, however structure in *point* modified so that point is snapped to grid.

◆ Objects

■ MSG_VIS_RULER_SNAP_TO_GRID_X

```
void MSG_VIS_RULER_SNAP_TO_GRID_X(
        PointDWFixed _far* point);
```

Snap the passed point's *x* coordinate to the grid.

Parameters: *point* Point to snap to grid.

Return: Nothing returned explicitly, however structure in *point* modified so that point is snapped to grid.

■ MSG_VIS_RULER_SNAP_TO_GRID_Y

```
void MSG_VIS_RULER_SNAP_TO_GRID_Y(
        PointDWFixed _far* point);
```

Snap the passed point's *y* coordinate to the grid.

Parameters: *point* Point to snap to grid.

Return: Nothing returned explicitly, however structure in *point* modified so that point is snapped to grid.

■ MSG_VIS_RULER_SNAP_RELATIVE_TO_REFERENCE

```
void MSG_VIS_RULER_SNAP_RELATIVE_TO_REFERENCE(
        PointDWFixed _far* point);
```

Returns the point closest to the passed point that is an integral number of grid spacings from the reference point.

Parameters: *point* Point to snap.

Return: Nothing returned explicitly, however structure in *point* modified so that point is snapped.

■ MSG_VIS_RULER_SNAP_RELATIVE_TO_REFERENCE_X

```
void MSG_VIS_RULER_SNAP_RELATIVE_TO_REFERENCE_X(
        PointDWFixed _far* point);
```

Returns the point horizontally closest to the passed point that is an integral number of grid spacings from the reference point.

Parameters: *point* Point to snap.

Return: Nothing returned explicitly, however structure in *point* modified so that point is snapped.

19.4

■ MSG_VIS_RULER_SNAP_RELATIVE_TO_REFERENCE_Y

```
void MSG_VIS_RULER_SNAP_RELATIVE_TO_REFERENCE_Y(
          PointDWFixed _far* point);
```

Returns the point vertically closest to the passed point that is an integral number of grid spacings from the reference point.

Parameters: *point* Point to snap.

Return: Nothing returned explicitly, however structure in *point* modified so that point is snapped.

19.4.4 Guide Constraints and Guidelines

Sometimes the user will want to have some guidelines that do not conform to a regular grid. The user may set up their own guidelines. Normally to allow the use of guidelines, your application should include a RulerGuideControl object. The features structures for this controller class are shown below.

Code Display 19-6 RulerGuideControlClass Features

```
typedef WordFlags RulerGuideControlFeatures;
#define RGCF_HV         (0x08)
#define RGCF_LIST       (0x04)
#define RGCF_POSITION   (0x02)
#define RGCF_DELETE     (0x01)

#define RULER_GUIDE_CONTROL_DEFAULT_FEATURES \
                        (RGCF_HV | RGCF_LIST | RGCF_POSITION | RGCF_DELETE)
```

The following messages allow for the use and manipulation of guidelines.

■ MSG_VIS_RULER_TURN_GUIDES_SNAPPING_ON

```
void MSG_VIS_RULER_TURN_GUIDES_SNAPPING_ON();
```

Set the default VisRulerConstrainStrategy to include guides snapping, turning on the VRCS_SNAP_TO_GUIDES_X and VRCS_SNAP_TO_GUIDES_Y flags.

■ MSG_VIS_RULER_TURN_GUIDES_SNAPPING_OFF

```
void MSG_VIS_RULER_TURN_GUIDES_SNAPPING_OFF();
```

Set the default VisRulerConstrainStrategy to not include guides snapping, turning off the VRCS_SNAP_TO_GUIDES_X and VRCS_SNAP_TO_GUIDES_Y flags.

■ MSG_VIS_RULER_CREATE_GUIDE_ARRAY

```
void MSG_VIS_RULER_CREATE_GUIDE_ARRAY();
```

By default, the VisRuler stores its guides in a chunk array, known as a guide array. If the ruler has no guides defined, it doesn't bother keeping this data structure around. When the first guide is created, the VisRuler will send itself this message, to which it will respond by setting up the data region.

Subclasses might intercept this message if they have data structures which need initializing when the first guide is created.

■ MSG_VIS_RULER_ADD_HORIZONTAL_GUIDE

```
void MSG_VIS_RULER_ADD_HORIZONTAL_GUIDE( @stack
        sdword              dwfInt,
        word                dwfFrac);
```

Use this message to add a horizontal guide.

■ MSG_VIS_RULER_ADD_VERTICAL_GUIDE

```
void MSG_VIS_RULER_ADD_VERTICAL_GUIDE( @stack
        sdword              dwfInt,
        word                dwfFrac);
```

Use this message to add a vertical guide.

■ MSG_VIS_RULER_DRAW_GUIDES

```
void MSG_VIS_RULER_DRAW_GUIDES(
        GStateHandle gstate);
```

Use this message to draw guidelines to a document. The default VisRuler handler for this message will draw the lines at the appropriate coordinates to the passed GState.

■ MSG_VIS_RULER_DRAW_GUIDE_INDICATORS

```
void MSG_VIS_RULER_DRAW_GUIDE_INDICATORS();
```

This message asks the VisRuler to draw tick-marks on itself to show the positions of guides. The default VisRuler has no handler for this message; subclasses are welcome to intersect it and draw indicators appropriately.

■ MSG_VIS_RULER_SNAP_TO_GUIDES

```
void MSG_VIS_RULER_SNAP_TO_GUIDES(
        PointDWFixed _far* point);
```

Use this message to snap the passed point to the guides.

■ MSG_VIS_RULER_SNAP_TO_GUIDES_X

```
void MSG_VIS_RULER_SNAP_TO_GUIDES_X(
        PointDWFixed _far* point);
```

Use this message to horizontally snap the passed point to the guides.

■ MSG_VIS_RULER_SNAP_TO_GUIDES_Y

```
void MSG_VIS_RULER_SNAP_TO_GUIDES_Y(
        PointDWFixed _far* point);
```

Use this message to vertically snap the passed point to the guides.

■ MSG_VIS_RULER_GET_GUIDE_INFLUENCE

```
word MSG_VIS_RULER_GET_GUIDE_INFLUENCE();
```

Guidelines will only affect the mouse if the mouse pointer lies within the guide's area of influence. This message returns the present influence value.

■ MSG_VIS_RULER_SET_GUIDE_INFLUENCE

```
void MSG_VIS_RULER_SET_GUIDE_INFLUENCE(
        word influence);
```

Guidelines will only affect the mouse if the mouse pointer lies within the guide's area of influence. This message sets the influence distance, measured in pixels.

■ MSG_VIS_RULER_SELECT_HORIZONTAL_GUIDE

```
void MSG_VIS_RULER_SELECT_HORIZONTAL_GUIDE( @stack
          sdword              dwfInt,
          word                dwfFrac);
```

Use this message to select a horizontal guide at the passed coordinates, if any. Note that only one horizontal guide may be selected at any time.

■ MSG_VIS_RULER_SELECT_VERTICAL_GUIDE

```
void MSG_VIS_RULER_SELECT_VERTICAL_GUIDE( @stack
          sdword              dwfInt,
          word                dwfFrac);
```

Use this message to select a vertical guide at the passed coordinates, if any. Note that only one vertical guide may be selected at any time.

■ MSG_VIS_RULER_DESELECT_ALL_HORIZONTAL_GUIDES

```
void MSG_VIS_RULER_DESELECT_ALL_HORIZONTAL_GUIDES();
```

Use this message to deselect all of the ruler's horizontal guides.

■ MSG_VIS_RULER_DESELECT_ALL_VERTICAL_GUIDES

```
void MSG_VIS_RULER_DESELECT_ALL_VERTICAL_GUIDES();
```

Use this message to deselect all of the ruler's vertical guides.

■ MSG_VIS_RULER_DELETE_HORIZONTAL_GUIDE

```
void MSG_VIS_RULER_DELETE_HORIZONTAL_GUIDE( @stack
          sdword              dwfInt,
          word                dwfFrac);
```

Use this message to delete the horizontal guide at the passed coordinate, if any.

■ MSG_VIS_RULER_DELETE_VERTICAL_GUIDE

```
void MSG_VIS_RULER_DELETE_VERTICAL_GUIDE( @stack
          sdword              dwfInt,
          word                dwfFrac);
```

Use this message to delete the vertical guide at the passed coordinate, if any.

19.4.5 Other Mouse Constraints

Up until now, we have discussed various ways to constrain mouse movement to grid- or guide-lines. However, other constraints are possible, and may be accessed through the following messages.

■ MSG_VIS_RULER_CONSTRAIN_TO_AXES

19.4

```
void MSG_VIS_RULER_CONSTRAIN_TO_AXES(
          PointDWFixed _far* point);
```

Use this message to constrain the mouse to the axes; that is to force it to be in line either horizontally or vertically with the reference point (the point stored in *VRI_reference*).

■ MSG_VIS_RULER_CONSTRAIN_TO_HORIZONTAL_AXIS

```
void MSG_VIS_RULER_CONSTRAIN_TO_HORIZONTAL_AXIS(
          PointDWFixed _far* point);
```

Use this message to constrain the mouse horizontally, forcing it to be in line horizontally with the reference point (the point stored in *VRI_reference*).

■ MSG_VIS_RULER_CONSTRAIN_TO_VERTICAL_AXIS

```
void MSG_VIS_RULER_CONSTRAIN_TO_VERTICAL_AXIS(
          PointDWFixed _far* point);
```

Use this message to constrain the mouse vertically, forcing it to be in line vertically with the reference point (the point stored in *VRI_reference*).

■ MSG_VIS_RULER_CONSTRAIN_TO_DIAGONALS

```
void MSG_VIS_RULER_CONSTRAIN_TO_DIAGONALS(
          PointDWFixed _far* point);
```

Use this message to constrain the mouse to the diagonal axes; that is to force it to be on a line 45 degrees through the reference point (the point stored in *VRI_reference*).

■ MSG_VIS_RULER_SET_CONSTRAIN_TRANSFORM

```
void MSG_VIS_RULER_SET_CONSTRAIN_TRANSFORM(
          TransMatrix* tm)
```

Use this message to impose a transformation on the coordinate system used to constrain the mouse. You can see this at work in GeoDraw when resizing an object which has been rotated or skewed.

■ MSG_VIS_RULER_CLEAR_CONSTRAIN_TRANSFORM

```
void MSG_VIS_RULER_CLEAR_CONSTRAIN_TRANSFORM();
```

Use this message to remove any transformation on the coordinate system used to constrain the mouse.

■ MSG_VIS_RULER_CONSTRAIN_X_TO_UNITY_SLOPE_AXIS

```
void MSG_VIS_RULER_CONSTRAIN_X_TO_UNITY_SLOPE_AXIS(
          PointDWFixed _far* point);
```

Use this message to determine the coordinates resulting from horizontally snapping a given point to the Northeast diagonal line passing through the reference point.

■ MSG_VIS_RULER_CONSTRAIN_Y_TO_UNITY_SLOPE_AXIS

```
void MSG_VIS_RULER_CONSTRAIN_Y_TO_UNITY_SLOPE_AXIS(
          PointDWFixed _far* point);
```

Use this message to determine the coordinates resulting from vertically snapping a given point to the Northeast diagonal line passing through the reference point.

■ MSG_VIS_RULER_CONSTRAIN_X_TO_NEGATIVE_SLOPE_AXIS

```
void MSG_VIS_RULER_CONSTRAIN_X_TO_NEGATIVE_SLOPE_AXIS(
          PointDWFixed _far* point);
```

Use this message to determine the coordinates resulting from horizontally snapping a given point to the Southeast diagonal line passing through the reference point.

■ MSG_VIS_RULER_CONSTRAIN_Y_TO_NEGATIVE_SLOPE_AXIS

```
void MSG_VIS_RULER_CONSTRAIN_Y_TO_NEGATIVE_SLOPE_AXIS(
         PointDWFixed _far* point);
```

Use this message to determine the coordinates resulting from horizontally snapping a given point to the Southeast diagonal line passing through the reference point.

■ MSG_VIS_RULER_CONSTRAIN_TO_VECTOR

```
@message void MSG_VIS_RULER_CONSTRAIN_TO_VECTOR(
         PointDWFixed _far* point);
```

This message computes the point resulting from constraining the passed point using vector constraint. This means that the mouse's constrained location will end up at a certain angle from the reference angle. This angle is determined by means of a point set up my MSG_VIS_RULER_SET_VECTOR. This constraint will constrain the point either to the vector line or to that line's reflection.

■ MSG_VIS_RULER_CONSTRAIN_X_TO_VECTOR

```
@message void MSG_VIS_RULER_CONSTRAIN_X_TO_VECTOR(
         PointDWFixed _far* point);
```

This message computes the point resulting from horizontally constraining the passed point to the line going through the reference point with the present vector slope.

■ MSG_VIS_RULER_CONSTRAIN_X_TO_VECTOR_REFLECTION

```
@message void MSG_VIS_RULER_CONSTRAIN_X_TO_VECTOR_REFLECTION(
         PointDWFixed _far* point);
```

This message computes the point resulting from horizontally constraining the passed point to the line going through the reference point with the negative of the present vector slope.

■ MSG_VIS_RULER_CONSTRAIN_Y_TO_VECTOR

```
@message void MSG_VIS_RULER_CONSTRAIN_Y_TO_VECTOR(
         PointDWFixed _far* point);
```

This message computes the point resulting from vertically constraining the passed point to the line going through the reference point with the present vector slope.

◆Objects

■ MSG_VIS_RULER_CONSTRAIN_Y_TO_VECTOR_REFLECTION

```
@message void MSG_VIS_RULER_CONSTRAIN_Y_TO_VECTOR_REFLECTION(
         PointDWFixed _far* point);
```

This message computes the point resulting from vertically constraining the passed point to the line going through the reference point with the negative of the present vector slope.

19.4.6 Esoteric Messages

For the most part, the ruler will work together with the main view to coordinate updates. This is done by means of messages going to the ruler. Subclasses of VisRulerClass might conceivably intercept these messages to alter behavior.

■ MSG_VIS_RULER_GAINED_SELECTION

```
void MSG_VIS_RULER_GAINED_SELECTION(
         optr               dest);
```

This message notifies the ruler that the ruled object (the object in the main view) is selected and the ruler should update its UI to reflect its own attributes.

Parameters: *dest* Object to send messages to whenever a change in ruler settings should result in the ruled object being redrawn (e.g. when the user wants to draw the grid).

Return: Nothing.

■ MSG_VIS_RULER_LOST_SELECTION

```
void MSG_VIS_RULER_LOST_SELECTION();
```

This message notifies the ruler that the ruled object is no longer selected.

■ MSG_VIS_RULER_UPDATE_CONTROLLERS

```
void MSG_VIS_RULER_UPDATE_CONTROLLERS();
```

This message signals that the ruler should update its associated controllers.

■ MSG_VIS_RULER_UPDATE_TYPE_CONTROLLER

```
void MSG_VIS_RULER_UPDATE_TYPE_CONTROLLER();
```

This message signals that the ruler should update its type controller, if any.

■ MSG_VIS_RULER_UPDATE_GRID_CONTROLLER

```
void MSG_VIS_RULER_UPDATE_GRID_CONTROLLER();
```

This message signals that the ruler should update its grid controller, if any.

■ MSG_VIS_RULER_UPDATE_GUIDE_CONTROLLER

```
void MSG_VIS_RULER_UPDATE_GUIDE_CONTROLLER();
```

This message signals that the ruler should update its guide controller, if any.

■ MSG_VIS_RULER_COMBINE_GUIDE_INFO

```
void MSG_VIS_RULER_COMBINE_GUIDE_INFO(
        MemHandle data);    /* Handle of block containing
                               VisRulerNotifyGuideChangeBlockHeader*/
```

This message allows the horizontal and vertical rulers to load their respective guide information into a single structure so that the information can be collated and passed on to a RulerGuideControl.

Structures:

```
typedef struct {
      LMemBlockHeader   VRNGCBH_header;
      word              VRNGCBH_vertGuideArray;
      word              VRNGCBH_horizGuideArray;
} VisRulerNotifyGuideChangeBlockHeader;
```

■ MSG_VIS_RULER_VIEW_SCALE_FACTOR_CHANGED

```
void MSG_VIS_RULER_VIEW_SCALE_FACTOR_CHANGED( @stack
        WindowHandle        viewWindow,
        WWFixedAsDWord      yScaleFactor,
        WWFixedAsDWord      xScaleFactor);
```

The ruler will receive this message when the main view's scale factor changes. Specifically, when the RulerContent 's scale is changing (which will happen because of the RulerView's link to the main view), its handler for MSG_META_CONTENT_VIEW_SCALE_FACTOR_CHANGED sends this message to the VisRuler.

◆ Objects

■ MSG_VIS_RULER_GET_DESIRED_SIZE

```
@message word MSG_VIS_RULER_GET_DESIRED_SIZE();
```

This message returns how much space the ruler thinks it needs to draw the numbers and hatch marks.

Interception: This method should be subclassed if the subclassed ruler's size needs to vary with respect to scale factor, whether the ruler is horizontal or vertical, or whatever size seems appropriate.

■ MSG_VIS_RULER_INVALIDATE_WITH_SLAVES

```
void MSG_VIS_RULER_INVALIDATE_WITH_SLAVES();
```

The ruler's default behavior will result in it redrawing itself at appropriate times. Depending on what use you put your ruler to, you may have to force it to redraw itself. If so, you should probably use this message.

19.4

Objects

Spreadsheet Objects

20

The spreadsheet objects provide most of the functionality that any spreadsheet will need. They let the user enter information in cells without direct involvement by the application. They also manage all interaction with the Parse library. The application can then ignore these issues, or it can intervene to add functionality.

Before you read this chapter, you should be familiar with the database routines (see “Database Library,” Chapter 19 of the Concepts Book). In particular, you should be familiar with the section on the cell library. You should also have read the first few sections of the Parse Library chapter (“Parse Library,” Chapter 20 of the Concepts Book); you need not read the “Advanced Usage” section. You should also be familiar with Ruler objects (“Ruler Object Library,” Chapter 19) and VisComp objects (“VisComp,” Chapter 24).

20.1 Spreadsheet Overview

The spreadsheet library provides a high-level interface with the Cell and Parse libraries. It also provides all the user interface for a spreadsheet. The objects are subclassed from common Vis objects and inherit much of their functionality from them; this means that applications can alter spreadsheet objects in a very intuitive way. Applications can also use the spreadsheet objects in “engine” mode. When the Spreadsheet is in engine mode, it does not provide any user interface, but it still provides a convenient interface to the Cell and Parse libraries.

There are three main spreadsheet objects; most spreadsheet applications will use all three. Applications which use the spreadsheet in engine mode will not need to use all of the objects. There are also several controllers which work with the spreadsheet objects. Most spreadsheet applications will end up including all of these.

20.1.1 Quick Look at the Objects

Three different classes of objects are provided for use with the spreadsheet. Most spreadsheet applications will use all three of these. To see how the objects are arranged, see Figure 20-1 on page ◆ 1226

SpreadsheetClass

This is the main spreadsheet object, a subclass of **VisCompClass**. The spreadsheet object manages a cell file; it automatically looks up values for cells and displays them, and copies new data into the cell file. This object does the bulk of a spreadsheet application's work. Every open spreadsheet will have its own Spreadsheet object; applications usually use the Document Control to duplicate this object automatically for every open file. All references in the text to the "Spreadsheet Object" are to this object.

SpreadSheetRuler

This is a subclass of **VisRulerClass**. It is usually placed just to the left of the Spreadsheet object. It usually contains the labels for the cell rows; however, it can be set to show measurements in any of the standard forms (inches, centimeters, points, etc.). Like all rulers, it is usually displayed in a RulerView and is usually the Vis child of a RulerContent. It is linked to the Spreadsheet object, and thus it can automatically show information about the active area of the spreadsheet. Most applications will have two rulers, one horizontal and one vertical (determined by settings of the VRA_HORIZONTAL flag) One ruler will receive messages from the Spreadsheet; the other ruler will be its slave. For more information about rulers, see "Ruler Object Library," Chapter 19.

SSEditBarControlClass

This is a subclass of **GenControlClass**. The controller displays the value or formula for the active cell. The user can use this controller to change the value in a cell. The controller is usually made a child of the Edit menu, if there is one. As with all controllers, the user can put this controller wherever he prefers; it is usually put at the top of the primary window, just below the menu bar.

Other Controllers

The Spreadsheet library provides other controllers which do such things as add notes to cells, change row and column width, and sort rows or columns. Applications which include these objects will not have to perform these tasks directly.

20.1.2 Managing Cell Files

As explained in “Database Library,” Chapter 19 of the Concepts Book, cell files are ways of accessing DB items in a VM file. Ordinarily, every DB item is accessed via two tokens: the VM handle of the item’s DB group, and the DB handle of the item. When items are arranged in a cell file, the application can access the items by a row and column address. In order to do this, the application must do some bookkeeping; for example, the application has to keep track of a **CellFunctionParameters** structure. If you use the spreadsheet objects, you can avoid this bookkeeping.

When you create a new spreadsheet file, the Spreadsheet object will allocate a “spreadsheet map” block in the file. (This is different from the ordinary VM map block; a file can have both a map block and a “spreadsheet map” block.) The spreadsheet objects will record essential bookkeeping information (such as the **CellFunctionParameters** structure) in this block. While it is running, it will cache the information in its own instance data; however, whenever the file is saved or updated, it will write the cache back to this block. The application should treat this block as entirely opaque.

Warning

You should not keep ungrouped DB items locked.

When the spreadsheet object attaches a new file, it must be passed the handle of the spreadsheet map block. It can then read all the information it needs about the file. The spreadsheet will automatically allocate cells when needed, and free them when appropriate. Note that since cells are ungrouped DB items, whenever a cell is allocated or resized, potentially any or all ungrouped items in that VM file could be moved. For this reason, if you use ungrouped items, you should not leave them locked; it’s probably safest not to use ungrouped items at all, if you can avoid them. (See section 19.2.4 of “Database Library,” Chapter 19 of the Concepts Book.)

Every cell in a spreadsheet can contain one of three possible values: a number, a text string, or a token string. The application may set a cell’s value with a message, or the user may set the selected cell’s value with the SSEditBarControl. If the user entered a number in any acceptable format, the spreadsheet will set the cell’s value to equal that number. If the user entered a character sequence which begins with an equals sign, the spreadsheet will call the parser to parse all the characters after the equals sign. If the sequence is a well-formed expression, the token sequence will be stored in the cell; otherwise an error will be displayed. If the user enters a character sequence which is not a well-formed number and does not begin with an equal sign, the spreadsheet object will interpret it as a string, and copy the string verbatim to the cell.

In addition, every cell can have a note associated with it. The note may be any null-terminated string. The note does not affect how the cell is evaluated. Users will generally add and view the notes with the SSNoteControl object.

Whenever the Spreadsheet changes any data in the file, it marks the appropriate block as dirty. If you use the Document Control objects, they will automatically take appropriate actions (such as enabling the "Save" trigger).

20.1.3 Parsing Expressions

The Spreadsheet object automatically calls the appropriate Parse Library routines to parse formula strings, as well as to evaluate and format token strings. It performs all necessary interaction with the Parse library; for example, when the Evaluator needs to know the current value of a cell, the Spreadsheet find's the cell's contents, performs any evaluations necessary, and returns the data to the Evaluator.

When the user enters a formula into a cell (i.e. a character sequence which begins with an equals sign), the spreadsheet automatically calls the Parse library routine to transform the character sequence into the corresponding token sequence. It stores the token sequence in the cell. It then calls the evaluator to simplify the token sequence. Finally, it calls the formatter to transform the simplified token sequence into a character string, and it displays this string in the spreadsheet display area. If the formula is well-formed, but cannot be evaluated (e.g. because it depends on a cell which has a bad value), it will display an appropriate error message in the cell. The formula is reevaluated whenever a the value of a cell or variable in the formula changes.

If the user wants to see or edit the actual formula, he can select the cell. The spreadsheet will call the formatter to transform the formula's token sequence back into a character string, which will be displayed in the edit bar, where the user can edit it. When the user is done editing, the new formula will again be parsed, and the new token sequence will be entered in the cell.

The Spreadsheet detects circular references; it can be instructed to break circular references, or to evaluate a set number of times through the circle to approximate the "true" value (i.e. the value if the circuit were followed an infinite number of times). Of course, this iteration is only useful if the cells' values converge.

20.2 The Spreadsheet Objects

There are four basic objects in the Spreadsheet library. These objects are fairly simple to use. Most of the functionality is in the Spreadsheet object; the other objects are straightforward modifications of classes which are documented elsewhere.

20.2.1 SpreadsheetClass

The spreadsheet object is the basis of the spreadsheet library. It does all the interaction with the cell and parse libraries. The spreadsheet has a lot of instance data fields. Some of them are of interest to the application; some are initialized by the application, and ignored thereafter; and some are of interest only to the spreadsheet object's internal code. The instance data is shown in Code Display 20-1. Those fields which are of interest to the application are described in detail in the following sections.

The spreadsheet object is a subclass of **VisCompClass**. If the spreadsheet is not being used in "engine" mode, it should be linked to a GenView object, just like any other VisComp object would be. For more information, "VisComp," Chapter 24.

Code Display 20-1 SpreadsheetClass Instance Data

```
/* This is the instance data for the SpreadsheetClass object. Some of these fields
 * must be initialized by the application; some are set and maintained entirely by
 * the object. */

/* The first field of the spreadsheet object is always the CellFunctionParameters
 * structure for the cell file. Therefore, a pointer to the instance data of a
 * Spreadsheet object can be used as a pointer to the CellFunctionParameters. */
    @instance CellFunctionParameters     SSI_cellParams= {0, 0, {0}};

/* SSI_maxRow and SSI_maxCol store the indices of the leftmost row and bottommost
 * column allowed in the spreadsheet. The application may set these; they must be
 * no larger than the spreadsheet library constants MAX_ROW (8191) and MAX_COLUMN
 * (255). */
    @instance word      SSI_maxRow = MAX_ROW;
    @instance word      SSI_maxColumn = MAX_COLUMN;

/* SSI_mapBlock is the VM handle of the attached file’s spreadsheet map block. The
 * spreadsheet sets this when a spreadsheet file is initialized; applications must
 * provide this when they instruct a spreadsheet object to open a file. */
    @instance VMBlockHandle SSI_mapBlock;

/* SSI_flags is a word-length record of SpreadsheetFlags. Some of these flags are
 * set by the application; others are set automatically by the spreadsheet object.
 * The application can change some of these at run-time. The SpreadsheetFlags are
 * described at length in section 20.2.1.4 on page 1218. The following flags are
 * available:
 *             SF_MANUAL_RECALC                SF_ALLOW_ITERATION
 *             SF_SUPPRESS_REDRAW              SF_APPLICATION_FUNCTIONS
 *             SF_QUICK_DRAW_IN_PROGRESS       SF_DOING_FEEDBACK
 *             SF_IN_VIEW                      SF_IS_TARGET
 *             SF_IS_FOCUS */
    @instance SpreadsheetFlags  SSI_flags = 0;

/* SSI_drawFlags is a word-sized record of SpreadsheetDrawFlags. These flags
 * specify how the spreadsheet will draw itself on-screen. The application can
 * change some of these at run-time. The SpreadsheetDrawFlags are described at
 * length in section 20.2.1.5 on page 1221. The following flags are available:
 *             SDF_DRAW_GRAPHICS       SDF_DRAW_NOTE_BUTTON
 *             SDF_DRAW_GRID           SDF_SIDEWAYS
 *             SDF_SCALE_TO_FIT        SDF_DRAW_ROW_COLUMN_TITLES
 *             SDF_SKIP_DRAW           SDF_CENTER_VERTICALLY
 *             SDF_CENTER_HORIZONTALLY SDF_CONTINUOUS
 *             SDF_DRAW_HEADER         SDF_DRAW_FOOTER
 *              SDF_PRINT_DOCUMENT       SDF_PRINT_NOTES
 *              SDF_DONE
 */
    @instance SpreadsheetDrawFlagsSSI_drawFlags = 0;

/* SSI_attributes is a byte-sized record of SpreadsheetAttributes flags. The flags
 * are discussed at length in section 20.2.1.6 on page 1224. The following flags
 * are available:
 *              SA_TARGETABLE            SA_ENGINE_MODE */
    @instance SpreadsheetAttributes      SSI_attributes = 0;

/* SSI_circCount specifies how many times the Spreadsheet should follow circular
 * references to approximate the cells' values. This is ignored if the
 * SF_ALLOW_ITERATION bit is not set. If the count is set to zero, circular
 * references will be flagged as errors. */
    @instance word      SSI_circCount = 0;

/* SSI_converge specifies how close an approximation we want with circular
 * references. If each of the cells in a circular reference changes its value by
 * less than the SSI_converge value, the spreadsheet will stop following the loop.
 * If the converge value is left at zero, the spreadsheet will make the full
 * SSI_circCount circuits. */
    @instance FloatNum  SSI_converge = {0, 0, 0, 0, 0};

/* SSI_ruler must be set by the application when the objects are defined. This
 * field holds an optr to one of the two SpreadsheetRuler objects; the other ruler
 * should be a slave of this ruler. */
    @instance optr      SSI_ruler;

/* The following instance data fields are for internal use by the spreadsheet.
 * Applications should never set or change these fields, and should not count on
 * values in these fields being consistent at any given time.*/

/* SSI_active contains a reference to the active cell. This field is automatically
 * set and maintained by the spreadsheet object. */
    @instance CellReference     SSI_active;

/* SSI_header and SSI_footer specify "header" and "footer" ranges of cells. These
 * cells will automatically be printed at the top and bottom of each page (when
 * the spreadsheet is printed). Users will generally set these with the Header and
 * Footer controllers. */
    @instance CellRange SSI_header = {0, 0, 0, 0};
    @instance CellRange SSI_footer = {0, 0, 0, 0};
```

```
/* The spreadsheet object caches the CellFunctionParameters structure in its own
 * instance data, and writes it to the spreadsheet map block whenever the file is
 * saved or updated. The spreadsheet object initializes and updates this field
 * automatically. */
    @instance CellFunctionParameters SSI_cellParams = {0, 0, {0}};

/*
 * The following fields are intended to be entirely transparent to the
 * application. Applications should not examine, set, or change these fields.
 */

/* SSI_chunk is the chunk handle of the spreadsheet object. This is set and
 * maintained by the object. */
    @instance ChunkHandle SSI_chunk = NullHandle;

/* SSI_styleArray, SSI_nameArray, SSI_formatArray, and SSI_nameArray store the
 * handles of VM blocks within the spreadsheet file. These blocks contain
 * information about the file. The handles are also stored in the spreadsheet map
 * block. The spreadsheet object sets and maintains these fields automatically. */
    @instance VMBlockHandle SSI_styleArray;
    @instance VMBlockHandle SSI_rowArray;
    @instance VMBlockHandle SSI_formatArray;
    @instance VMBlockHandle SSI_nameArray;

/* SSI_offset is the offset to the visible portion of the spreadsheet. This field
 * is maintained automatically by the spreadsheet object. */
    @instance PointDWord SSI_offset;

/* SSI_offset stores the row and column indices of the currently visible portion
 * of the spreadsheet. This field is maintained automatically by the spreadsheet
 * object. */
    @instance CellRange  SSI_visible;

/* SSI_quickSource is used for bookkeeping during a quick move/copy. During the
 * move, the row/column indices of the source area are stored here. */
    @instance CellRange  SSI_quickSource;

/* SSI_selected contains the row/column indices of the currently selected region.
 * This field is automatically set and maintained by the spreadsheet object. */
    @instance CellRange SSI_selected;

/* SSI_curAttrs contains the token for the current style attributes. All text
 * entered will have this style. This field is automatically set and maintained by
 * the spreadsheet object. */
    @instance word      SSI_curAttrs;

/* SSI_gstate is used to cache the gstate handle used for drawing to the screen.
 * This field is set and maintained by the spreadsheet object. */
    @instance word      SSI_gstate;
```

```
/* SSI_gsRefCount is used to keep track of the number of references to the gstate;
 * this lets the spreadsheet object know when to free the gstate. This field is
 * set and maintained by the spreadsheet object. */
    @instance byte      SSI_gsRefCount;

/* SSI_ancestorList, SSI_childList, and SSI_finalList are cached in the instance
 * data for efficiency. Applications should ignore these fields. */
    @instance word      SSI_ancestorList;
    @instance word      SSI_childList;
    @instance word      SSI_finalList;

/* SSI_bounds holds the bounds of the portion of the spreadsheet which contains
 * data. This field is set and maintained by the spreadsheet. */
    @instance RectDWord SSI_bounds = {0, 0, 0, 0}
```

20.2

20.2.1.1 SSI_mapBlock

The spreadsheet object has to keep track of certain information about files between executions. To do this, it writes data in a “spreadsheet map block.” This block may be any VM block in the file containing the spreadsheet; it need not be the file’s map block (indeed, it usually is not). When the spreadsheet object initializes a new spreadsheet file, it allocates a map block and returns the spreadsheet map block’s VM handle. When you attach an existing file to the spreadsheet object (with MSG_SPREADSHEET_ATTACH_FILE), you must provide this handle; the spreadsheet will lock the map block and get all appropriate information from it. Applications may not access the spreadsheet map block directly; they should never lock or unlock it.

Initializing and attaching files is described at length in section 20.3.2 on page 1230.

20.2.1.2 SSI_maxRow and SSI_maxColumn

The application can specify the maximum dimensions for the spreadsheet. The upper-left corner of the spreadsheet is always cell A1 (row index 0, column index 0). The application can specify the maximum row and column indices allowed. These indices must be less than or equal to the constants MAX_ROW (8191) and MAX_COL (255).

20.2.1.3 The Active Cell

One of the cells in a spreadsheet is always the “active” cell. When a formula or value is entered in the formula bar, the value is copied to the active cell. Similarly, the formula bar displays the formula for the active cell (whereas the cell’s space in the spreadsheet displays the value to which the formula evaluates). The active cell’s coordinates are stored in the fields *SSI_curRow* and *SSI_curCol*. Ordinarily, the user—not the application—sets the active cell, not the application. However, the application can change the active cell by sending the message MSG_SPREADSHEET_MOVE_ACTIVE_CELL.

■ MSG_SPREADSHEET_MOVE_ACTIVE_CELL

```
void    MSG_SPREADSHEET_MOVE_ACTIVE_CELL(
        word   row,        /* Move to cell with this row index */
        word   column);    /* and this column index */
```

This message instructs the spreadsheet to change the active cell. Applications should not usually need to send this message; they can rely on the spreadsheet’s user interface for this.

Source: Unrestricted.

Destination: Any Spreadsheet object.

Parameters: *row* The row index of the new active cell. This should be specified in zero-based form; that is, the first row has an index of zero.

column The column index of new active cell. This should be specified in zero-based form; that is, the first column has an index of zero.

Interception: This message is not ordinarily subclassed.

20.2.1.4 Recalculation and Iteration

```
SpreadsheetFlags, SpreadsheetRecalcParams,
MSG_SPREADSHEET_CHANGE_RECALC_PARAMS,
MSG_SPREADSHEET_GET_RECALC_PARAMS, MSG_SPREADSHEET_RECALC
```

The Spreadsheet object uses the **SpreadsheetFlags** record to keep track of details about the Spreadsheet’s current state. Some of these fields are saved to the Spreadsheet map block, and some are not. There are only a two fields in this record that are of interest to the application:

SF_MANUAL_RECALC

If this flag is clear, whenever a cell's value is changed, the spreadsheet will automatically recalculate the cell's value (and the values of all cells which depend on it). If the flag is set, the application must force a recalculation by sending MSG_SPREADSHEET_RECALC to the spreadsheet object. By default, the flag is clear.

SF_ALLOW_ITERATION

If this flag is clear, the spreadsheet will not allow circular cell references (e.g. A1 = 10 + B1, B1 = 1/A1); all cells in a circular reference will evaluate to the error PSEE_CIRCULAR_REF, and all cells which depend on these cells will evaluate to PSEE_CIRCULAR_DEP. If the flag is set, the spreadsheet will follow circular references for a set number of circuits (specified in *SSI_circCount*) or until values converge within a specified tolerance (specified in *SSI_converge*).

Both of these flags are saved in the spreadsheet map block; thus, when a spreadsheet file is opened, these flags will be set to their values as of the last time the file was saved.

If you allow circular references, you must specify how many times a circuit will be followed. You do this by setting two instance data fields, *SSI_circCount* and *SSI_converge*. *SSI_circCount* specifies how many times a circuit should be followed in order to approximate the "true" values. If you set *SSI_circCount* to zero, the Spreadsheet will act the same as if SF_ALLOW_ITERATION had been cleared. You can also set a threshold value in the FloatNum datum *SSI_converge*. If, during a circuit, none of the cells' values change by more than the amount specified in *SSI_converge*, the spreadsheet will accept the values as accurate (even if it has not completed *SSI_circCount* circuits). If *SSI_converge* is set to zero, the spreadsheet will make the full *SSI_circCount* iterations.

To change the current recalculation parameters, send MSG_SPREADSHEET_CHANGE_RECALC_PARAMS to the spreadsheet object. This message takes one argument, a pointer to a **SpreadsheetRecalcParams** structure:

```
typedef struct {
      SpreadsheetFlags  SRP_flags;
      word              SRP_circCount;
      FloatNum          SRP_converge;
} SpreadsheetRecalcParams;
```

The spreadsheet object will set its *SSI_circCount* instance datum to equal the setting of *SRP_circCount*, and will set *SSI_converge* to equal *SRP_converge*. It will also set its SF_MANUAL_RECALC and SF_ALLOW_ITERATION flags to match their settings in *SRP_flags*; it will not change any of the other flags in *SSI_flags*.

To find out the current recalculation/iteration settings, call MSG_SPREADSHEET_GET_RECALC_PARAMS. This message takes one argument, namely a pointer to a **SpreadsheetRecalcParams** structure. The spreadsheet object will copy its *SSI_flags*, *SSI_circCount*, and *SSI_converge* data into the structure's *SRP_flags*, *SRP_circCount*, and *SRP_flags* fields.

■ MSG_SPREADSHEET_RECALC

void MSG_SPREADSHEET_RECALC()

This message instructs the spreadsheet to recalculate the values of all of its cells. This message is needed if the spreadsheet has had automatic recalculation disabled; otherwise, the spreadsheet will automatically recalculate cells as needed.

Source: Unrestricted.

Destination: Any Spreadsheet object.

Parameters: None.

Return: Nothing.

Interception: This message is not ordinarily subclassed.

■ MSG_SPREADSHEET_GET_RECALC_PARAMS

void MSG_SPREADSHEET_GET_RECALC_PARAMS(
SpreadsheetRecalcParams *ssRecalcParams);

This message gets the current values of the Spreadsheet object's *SSI_flags*, *SSI_circCount*, and *SSI_converge* fields. The fields are copied into the corresponding fields of a **SpreadsheetRecalcParams** structure.

Source: Unrestricted.

Destination: Any Spreadsheet object.

Parameters: *ssRecalcParams* A pointer to a **SpreadsheetRecalcParams** structure. The data will be copied into this structure.

Return: Appropriate values copied into **ssRecalcParams*.

Structures: **SpreadsheetRecalcParams** (described on page 1220).

Interception: This message is not ordinarily subclassed.

■ MSG_SPREADSHEET_CHANGE_RECALC_PARAMS

```
void        MSG_SPREADSHEET_CHANGE_RECALC_PARAMS(
            const SpreadsheetRecalcParams*    ssRecalcParams);
```

This message changes the current values of the spreadsheet object's *SSI_flags*, *SSI_circCount*, and *SSI_converge* fields. The spreadsheet object copies the **SpreadsheetRecalcParams** structure's *SRP_circCount*, and *SRP_converge* fields into its own *SSI_circCount* and *SSI_converge* fields. It also sets its SF_MANUAL_RECALC and SF_ALLOW_ITERATION flags to correspond to the settings in *SRP_flags*; it does not change any of the other flags in *SSI_flags*.

Source: Unrestricted.

Destination: Any Spreadsheet object.

Parameters: *ssRecalcParams* A pointer to a **SpreadsheetRecalcParams** structure. The settings will be copied from this structure into the spreadsheet's instance data.

Return: Nothing.

Structures: **SpreadsheetRecalcParams** (described on page 1220).

Interception: This message is not ordinarily subclassed.

20.2.1.5 SpreadsheetDrawFlags

`SpreadsheetDrawFlags, MSG_SPREADSHEET_GET_DRAW_FLAGS, MSG_SPREADSHEET_ALTER_DRAW_FLAGS`

The application can specify how the spreadsheet should draw itself by setting the **SpreadsheetDrawFlags**. Many of these flags are relevant only when the spreadsheet is being printed; they are ignored when the spreadsheet is being drawn to the screen. The application can set these in the object declaration, or it can change them at run-time by sending MSG_SPREADSHEET_ALTER_DRAW_FLAGS. It can also find out the current settings by sending MSG_SPREADSHEET_GET_DRAW_FLAGS.

SpreadsheetDrawFlags has the following flags:

SDF_DRAW_GRAPHICS
Draw graphics associated with the spreadsheet.

SDF_DRAW_NOTE_BUTTON
Add glyphs in cells which contain notes, as well as in header or footer cells.

SDF_DRAW_GRID
Draw grid lines between cells.

SDF_SIDEWAYS
Draw the spreadsheet sideways. This is relevant only for printing.

SDF_SCALE_TO_FIT
Scale the selected range to fill a page. This is significant only for printing.

SDF_DRAW_ROW_COLUMN_TITLES
Draw the row and column titles.

SDF_SKIP_DRAW
Do not draw anything to the screen. Set this if you want to find out how the pages will lie, without actually drawing or printing them.

SDF_CENTER_VERTICALLY
Center each page's contents vertically. This is relevant only for printing.

SDF_CENTER_HORIZONTALLY
Center each page's contents vertically. This is relevant only for printing.

SDF_CONTINUOUS
Print the spreadsheet continuously; that is, don't leave space between pages or at the edges (or leave as little margin as the printer allows). This is relevant only for printing.

SDF_DRAW_HEADER
Print the header cells at the top of each page. This is relevant only for printing.

SDF_DRAW_FOOTER
Print the footer cells at the bottom of each page. This is relevant only for printing.

SDF_PRINT_DOCUMENT
This flag is never set or checked by the spreadsheet object. It is provided for the convenience of objects which use the spreadsheet.

SDF_PRINT_NOTES
This flag is never set or checked by the spreadsheet object. It is provided for the convenience of objects which use the spreadsheet.

■ MSG_SPREADSHEET_GET_DRAW_FLAGS

```
SpreadsheetDrawFlags MSG_SPREADSHEET_GET_DRAW_FLAGS()
```

This message returns the spreadsheet object's *SSI_drawFlags* instance data field.

Source: Unrestricted.

Destination: Any Spreadsheet object.

Parameters: None.

Return: Spreadsheet object's *SSI_drawFlags* field.

Interception: This message is not ordinarily subclassed.

■ MSG_SPREADSHEET_ALTER_DRAW_FLAGS

```
void        MSG_SPREADSHEET_ALTER_DRAW_FLAGS(
            SpreadsheetDrawFlags    bitsToSet,
            SpreadsheetDrawFlags    bitsToClear);
```

This message changes the settings of the spreadsheet object's *SSI_drawFlags* field. If the same bit is set in *bitsToSet* and *bitsToClear*, the results are undefined.

Source: Unrestricted.

Destination: Any Spreadsheet object.

Parameters: *bitsToSet* Turn on these flags in *SSI_drawFlags*.

bitsToClear Turn off these flags in *SSI_drawFlags*.

Return: Nothing.

Interception: This message is not ordinarily subclassed.

20.2.1.6 Engine Mode

`SpreadsheetAttributes`

Some applications will want to use the spreadsheet object to manage cell files and to interact with the parser, but will not want to use its user-interface. These applications can run the spreadsheet in “engine” mode. When the spreadsheet is in engine mode, it does not draw anything to the screen. Applications must arrange for the spreadsheet object to get data for cells, and must directly request data from the cells.

To run a spreadsheet in engine mode, set the SA_ENGINE_MODE bit in the field *SSI_attributes*. This is the only field in *SSI_attributes* with which applications need concern themselves.

20.2.1.7 SSI_ruler

The spreadsheet object needs to be linked to the SpreadsheetRuler objects. To set up the link, put an optr to one of the rulers in *SSI_ruler*; have the other ruler be a slave of the first.

20.2.2 Spreadsheet Rulers

A special ruler object is defined for use with spreadsheets, **SpreadsheetRulerClass**. Applications can treat this the same way they treat ordinary ruler objects. Most applications will have two ruler objects, one horizontal and one vertical. Each of these should be a child of a RulerContent object, and should be displayed in a RulerView. For more information about rulers, see “Ruler Object Library,” Chapter 19.

Each ruler has a single instance datum which ordinary rulers lack, namely *SRI_spreadsheet*. This field should be initialized to contain an optr to the spreadsheet object.

20.2.3 The Spreadsheet Controller

Users enter data and formulae into the spreadsheet by using the Spreadsheet Edit-Bar Controller (i.e. **SSEditBarControlClass**, a subclass of **GenControlClass**). The controller is usually displayed in the tool-palette just below the menu bar; however, as with all controllers, the user can put the edit bar wherever he prefers.

The edit bar has two main functions: it is used to *display* the contents of cells, and it is used to *change* the contents. The spreadsheet object does not exactly display the contents of the cells. To be sure, if the cell contains a number or a string, the spreadsheet will display this in the cell's space; however, if the cell contains a formula, the spreadsheet will display not the formula, but rather the value to which the cell evaluates. For example, if a cell contains the formula "= 1 + 1", the spreadsheet object will display the number two in the cell. The edit bar, on the other hand, displays the contents of the active cell; in this case, it would display the text "= 1 + 1". 20.3

The edit bar is also used for changing the contents of cells. As noted, the edit bar displays the contents of the active cell. The user can change this text at will. The edit bar contains two special icons. One instructs the spreadsheet to store the new formula in the cell; this causes the edit bar to pass the string to the spreadsheet, which parses the string and stores the parsed version in the cell, then recalculates all affected cells. The other icon instructs the edit bar to throw out the changed version, and get the original formula from the spreadsheet. The edit bar also displays the coordinates of the active cell.

SSEditBarControlClass has no instance data (beyond what **GenControlClass** has). It uses only one message which is of interest to applications, namely MSG_SSEBC_INITIAL_KEYPRESS. This message is described below on page 1235.

20.3 Basic Use

The spreadsheet objects are very simple to use. Once the application defines the objects, they almost run themselves. There are a few messages which applications have to know to send; these are described below. There are also some advanced utilities which applications may want to use.

20.3.1 Declaring the Objects

The Spreadsheet objects are all subclassed off of fairly common objects, and should be declared the way those objects would be. The SSEditBarControl is generally the child of an “edit menu” interaction, and is often placed in the same resource as the edit menu; it should be run by the UI thread. The Spreadsheet and SpreadsheetRuler objects are usually put in the same resource. The rulers should be children of RulerContent objects. The spreadsheet object, as well as the ruler contents, should be displayed in their own GenView objects.

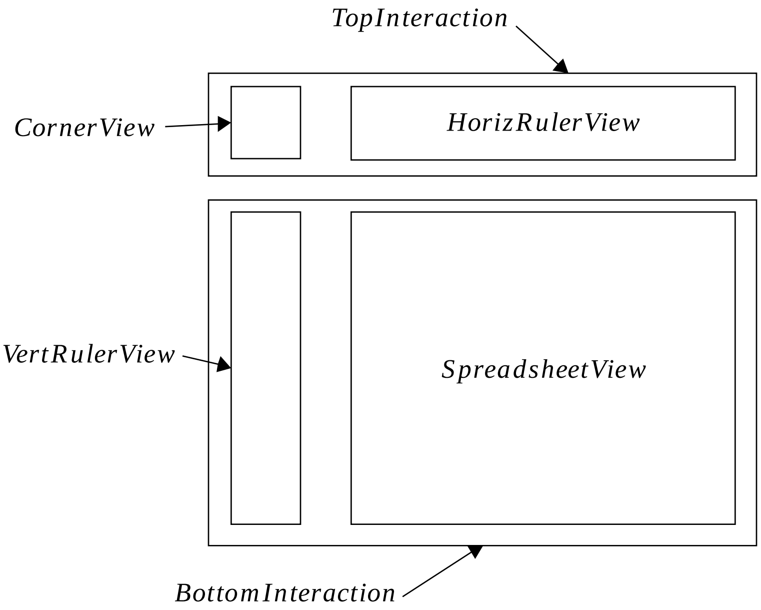


Figure 20-1 *Arranging Spreadsheet Objects*

This is how the sample spreadsheet objects are arranged on the page. Each Spreadsheet object (except for the controllers) is displayed in its own view. Applications will also have to declare a special view to space the objects properly. For an example of how the objects are declared, see Code Display 20-2 on page ◆ 1227.

A simple usage of the spreadsheet objects is shown in Code Display 20-2 on page ◆ 1227. The arrangement of objects can be confusing; Figure 20-1 on page ◆ 1226 is a diagram of the various GenViews and GenInteractions and how they are arranged.

Code Display 20-2 Declaring the Spreadsheet Objects

```
/* First, we declare the Generic UI objects. These will be run by the UI thread.
 * Not all of the application's objects are shown here. */

@start Interface;

/* The SSEditBarControl is the child of an edit menu. It is declared just like any
 * other controller. */
@object SampleEditBarControl SSEditBarControl = {
    GI_visMoniker = "Edit bar";
    GI_visibility = dialog;
    HINT_EXPAND_WIDTH_TO_FIT_PARENT;
    HINT_GEN_CONTROL_USE_DEFAULT_TOOLS;
}

@end Interface

/*****************************************************************************
 *      View Objects
 *****************************************************************************/

/* All of the views will be in a single resource. They will children of the
 * GenPrimary; GenInteraction objects are used to arrange the objects on the
 * screen. */

@start ViewResource;

/* This constant is used for sizing. */
#define RULER_WIDTH     40
#define DOCUMENT_WIDTH  MAX_COORD
#define DOCUMENT_HEIGHT MAX_COORD


/* This GenInteraction contains all the Spreadsheet view objects. */
@object GenInteractionClass SampleOuterInteraction = {
    GI_comp = TopInteraction, BottomInteraction;
    HINT_ORIENT_CHILDREN_VERTICALLY;
    HINT_MINIMIZE_CHILD_SPACING;
    HINT_EXPAND_WIDTH_TO_FIT_PARENT;
    HINT_EXPAND_HEIGHT_TO_FIT_PARENT;
}

/* This GenInteraction contains the horizontal ruler view, as well as a
 * space-filling corner view. */
@object GenInteractionClass TopInteraction = {
    GI_comp = CornerView, HorizRulerView;
    HINT_ORIENT_CHILDREN_HORIZONTALLY;
    HINT_MINIMIZE_CHILD_SPACING;
    HINT_EXPAND_WIDTH_TO_FIT_PARENT;
}

/* This GenInteraction contains the vertical ruler view and the Spreadsheet
 * object’s view. */
@object GenInteractionClass BottomInteraction = {
    GI_comp = VertRulerView, SpreadsheetView;
    HINT_ORIENT_CHILDREN_HORIZONTALLY;
    HINT_MINIMIZE_CHILD_SPACING;
    HINT_EXPAND_WIDTH_TO_FIT_PARENT;
    HINT_EXPAND_HEIGHT_TO_FIT_PARENT;
}

/* The CornerView is used to make the space where the horizontal and vertical
 * rulers meet. */
@object GenViewClass CornerView = {
    GI_attrs = @default & ~GA_TARGETABLE;
    GVI_attrs = @default | GVA_SAME_COLOR_AS_PARENT_WIN;
    GVI_horizAttrs = @default | GVDA_NO_LARGER_THAN_CONTENT;
    GVI_vertAttrs = @default | GVDA_NO_LARGER_THAN_CONTENT;
    ATTR_GEN_VIEW_PAGE_SIZE = {RULER_WIDTH, RULER_WIDTH};
    HINT_FIXED_SIZE = {HEADER_WIDTH, HEADER_WIDTH, 0};
}

/* These RulerViews are used to display the spreadsheet rulers. */

@object RulerViewClass HorizRulerView = {
    ATTR_GEN_VIEW_PAGE_SIZE = {DOCUMENT_WIDTH, RULER_WIDTH};
    HINT_FIXED_SIZE {0, HEADER_HEIGHT, 0};
    GVI_content = HorizRulerContent;
}

@object RulerViewClass VertRulerView = {
    ATTR_GEN_VIEW_PAGE_SIZE = {RULER_WIDTH, DOCUMENT_HEIGHT};
    HINT_FIXED_SIZE {0, HEADER_HEIGHT, 0};
    GVI_content = VertRulerContent;
}
```

Objects

```
/* The spreadsheet is displayed in its own view. The content of this view is the
 * spreadsheet object itself. As explained in section 20.3.3 on page 1234, this
 * view must have a special handler for MSG_META_KBD_CHAR; for this reason, we use
 * a subclass of GenViewClass. */
@object SampleSpreadsheetViewClass SpreadsheetView = {
    GVI_attrs = @default | GVA_CONTROLLED | GVA_DONT_SEND_POINTER_RELEASES
                | GVA_DRAG_SCROLLING | GVA_WINDOW_COORDINATE_MOUSE_EVENTS;

    GVI_horizAttrs = @default | GVDA_SCROLLABLE;
    GVI_vertAttrs = @default | GVDA_SCROLLABLE;
    ATTR_GEN_VIEW_PAGE_SIZE = {DOCUMENT_WIDTH, DOCUMENT_HEIGHT};
    HINT_DEFAULT_FOCUS;
    HINT_DEFAULT_TARGET;
    GVI_CONTENT = SampleSpreadsheet;
}

@end ViewResource

/***************************************************************************
 *      Vis Objects
 ***************************************************************************/

/* These are the actual spreadsheet objects, as well as the contents in which the
 * rulers are displayed. The spreadsheet object is itself a subclass of
 * VisContent, so it need not be a child of a content. This resource is generally
 * run by the application thread. */

@start VisResource;

/* The spreadsheet object can generally be left with its default settings. */
@object SpreadsheetClass SampleSpreadsheet = {
    VI_bounds = {0,0,DOCUMENT_WIDTH, DOCUMENT_HEIGHT};
    VI_attrs = @default & ~VA_MANAGED;
    VI_optFlags = @default & ~VOF_GEOMETRY_INVALID & ~VOF_GEO_UPDATE_PATH;
    SSI_drawFlags = SDF_DRAW_GRID | SDF_DRAW_NOTE_BUTTON;
    SSI_ruler = HorizRuler;
}

/* Each ruler object is the child of a ruler content. */
@object RulerContentClass HorizRulerContent = {
    VI_bounds = {0,0, DOCUMENT_WIDTH, RULER_WIDTH};
    VI_attrs = @default & ~VA_MANAGED;
    VI_optFlags = @default & ~VOF_GEOMETRY_INVALID & ~VOF_GEO_UPDATE_PATH;
    VCI_comp = HorizRuler;
    VCNI_attrs = VCNA_SAME_HEIGHT_AS_VIEW | VCNA_LARGE_DOCUMENT_MODEL \
                | VCNA_WINDOW_COORDINATE_MOUSE_EVENTS;
}
```

```
/* We declare the HorizRuler as the “master” ruler; the spreadsheet sends messages
 * to it, and it relays them to the “slave” VertRuler. (It could as easily have
 * been the other way around.) */
@object SpreadsheetRulerClass HorizRuler = {
    VRI_type = VRT_CUSTOM;
    VI_bounds = {0, 0, DOCUMENT_WIDTH, RULER_WIDTH};
    VI_attrs = @default & ~ VA_MANAGED;
    VI_optFlags = @default & ~VOF_GEOMETRY_INVALID & ~VOF_GEO_UPDATE_PATH;
    SRI_spreadsheet = SampleSpreadsheet;
    VRI_slave = VertRuler;
    VRI_rulerAttrs = @default & ~(VRA_SHOW_GRID | VRA_SHOW_MOUSE | VRA_HORIZONTAL);
}

/* This is the content object for the vertical ruler. */
@object RulerContentClass VertRulerContent = {
    VI_bounds = {0,0, RULER_WIDTH, DOCUMENT_HEIGHT};
    VI_attrs = @default & ~VA_MANAGED;
    VI_optFlags = @default & ~VOF_GEOMETRY_INVALID & ~VOF_GEO_UPDATE_PATH;
    VCI_comp = HorizRuler;
    VCNI_rulerAttrs = VCNA_SAME_WIDTH_AS_VIEW | VCNA_LARGE_DOCUMENT_MODEL \
                 | VCNA_WINDOW_COORDINATE_MOUSE_EVENTS;
}

/* We declare the VertRuler as the “slave” ruler; it will get its messages from
 * the “master” HorizRuler. (It could as easily have been the other way around.) */
@object SpreadsheetRulerClass VertRuler = {
    VRI_type = VRT_CUSTOM;
    VI_bounds = {0, 0, RULER_WIDTH, DOCUMENT_HEIGHT};
    VI_attrs = @default & ~ VA_MANAGED;
    VI_optFlags = @default & ~VOF_GEOMETRY_INVALID & ~VOF_GEO_UPDATE_PATH;
    SRI_spreadsheet = SampleSpreadsheet;
    VRI_rulerAttrs = \
        (@default & ~(VRA_SHOW_GRID | VRA_SHOW_MOUSE)) | VRA_HORIZONTAL;
}

@end VisResource
```

20.3.2 Working with Files

SpreadsheetInitFile(), SpreadsheetInitFileData, MSG_SPREADSHEET_ATTACH_FILE, MSG_SPREADSHEET_GET_FILE,

MSG_SPREADSHEET_READ_CACHED_DATA,
MSG_SPREADSHEET_WRITE_CACHED_DATA

The spreadsheet objects work with files almost transparently to the application. The application need only take a few steps to set things up and keep them working.

When an application creates a new spreadsheet file, it must first create a VM file, and then call **SpreadsheetInitFile()**. This routine initializes a cell file in the VM file; it also sets up the spreadsheet map block. It returns the handle of the spreadsheet map block; applications must store this, since they will need to pass it when they want to attach the file.

SpreadsheetInitFile() takes one argument: a pointer to a **SpreadsheetInitFileData** structure. **SpreadsheetInitFileData** has the following fields:

```
typedef struct {
      VMFileHandle      SIFD_file;
      word              SIFD_numRows;
      word              SIFD_numCols;
} SpreadsheetInitFileData;
```

SIFD_file This is the VM handle of the spreadsheet file.

SIFD_numRows
This is the number of rows in the spreadsheet. It is ordinarily equal to the spreadsheet's instance data field *SSI_maxRow* + 1.

SIFD_numCols
This is the number of columns in the spreadsheet. It is ordinarily equal to the spreadsheet's instance data field *SSI_maxCol* + 1.

When an application has just initialized a new spreadsheet file or opened a pre-existing one, it must attach the file to the spreadsheet object. The application can do this by sending MSG_SPREADSHEET_ATTACH_FILE to the spreadsheet object. This message takes two arguments: the VM file's handle, and the VMBlockHandle of the spreadsheet map block. The spreadsheet will copy appropriate information from the spreadsheet map block into its own instance data and will display the visible cells. You can find out what file is attached by sending MSG_SPREADSHEET_GET_FILE to the spreadsheet object; this message returns the file's handle.

When you save the VM file, you must make sure that the spreadsheet copies all of its cached data to the file. That way the spreadsheet will be able to get up-to-date information when it restarts. To do this, send the message MSG_SPREADSHEET_WRITE_CACHED_DATA to the spreadsheet object. This message takes one argument, namely the handle of the VM file. This may be a new file if, for example, you are performing a “save-as” operation; the spreadsheet will assume that the file handle you pass supersedes its existing file handle. Applications which use the Document Control objects should send this message in their handlers for MSG_GEN_DOCUMENT_WRITE_CACHED_DATA_TO_FILE (see “GenDocument,” Chapter 13).

At times, you may need to instruct the spreadsheet objects to reread its cached data from the file. For example, if you revert the file to its last-saved state, the spreadsheet will have to reread all its cached data. To do this, send MSG_SPREADSHEET_READ_CACHED_DATA to the spreadsheet object. This message takes two arguments: the file handle, and the spreadsheet map block’s **VMBlockHandle**. These supercede the handles stored in the spreadsheet’s instance data. The spreadsheet responds to this message by opening the specified map block and rereading its instance data from this block. Applications which use the Document Control objects should send this message in their handlers for MSG_GEN_DOCUMENT_READ_CACHED_DATA_FROM_FILE.

■ MSG_SPREADSHEET_ATTACH_FILE

```
void    MSG_SPREADSHEET_ATTACH_FILE(
        VMBlockHandle      mapBlockHandle,
        VMFileHandle       fileHandle);
```

This message instructs a spreadsheet object to attach itself to a spreadsheet file. The spreadsheet object will copy the appropriate information from the specified map block into its own instance data. Applications must send this when they open any spreadsheet file, whether a newly-initialized one or a pre-existing one.

Source: Unrestricted.

Destination: Any Spreadsheet object.

Parameters: *mapBlockHandle* The VMBlockHandle of the spreadsheet map block.

fileHandle The VMFileHandle of the spreadsheet file.

Interception: This message is not ordinarily subclassed.

■ MSG_SPREADSHEET_GET_FILE

```
VMFileHandle MSG_SPREADSHEET_GET_FILE()
```

This message returns the VMFileHandle of the file attached to the recipient Spreadsheet object.

Source: Unrestricted.

Destination: Any Spreadsheet object.

Parameters: None.

Return: The VMFileHandle of the attached file.

Interception: This message is not ordinarily subclassed.

■ MSG_SPREADSHEET_READ_CACHED_DATA

```
void        MSG_SPREADSHEET_READ_CACHED_DATA(
            VMFileHandle        fileHandle,
            VMBlockHandle       mapBlockHandle);
```

This message instructs a spreadsheet object to read its cached data from the specified file and block. The handles passed supersede the handles in the Spreadsheet object's instance data. Applications which use the Document Control should send this message in their handlers for MSG_GEN_DOCUMENT_READ_CACHED_DATA_FROM_FILE.

Source: Unrestricted.

Destination: Any Spreadsheet object.

Parameters: *fileHandle* The VMFileHandle of the spreadsheet file.

mapBlockHandle The VMBlockHandle of the spreadsheet map block.

Interception: This message is not ordinarily subclassed.

■ MSG_SPREADSHEET_WRITE_CACHED_DATA

```
void        MSG_SPREADSHEET_WRITE_CACHED_DATA(
            VMFileHandle        file);
```

This message instructs the spreadsheet object to write any cached data to the spreadsheet file. The file handle passed supersedes any file handle in the spreadsheet's instance data. Applications which use the Document Control should send this message in their handlers for MSG_GEN_DOCUMENT_READ_CACHED_DATA_FROM_FILE.

Source: Unrestricted.

Destination: Any Spreadsheet object.

Parameters: *fileHandle* The VMFileHandle of the spreadsheet file.

Interception: This message is not ordinarily subclassed.

20.3.3 Interacting with the Edit Bar

20.3

The spreadsheet and the edit bar will work together with very little intervention from the application. There is only one task which the application needs see to personally.

When the user's focus is on the spreadsheet, some of the keypresses will be for navigation; for example, the arrow keys change the active cell. However, some keypresses will be intended for the formula bar. For example, if the focus is on the spreadsheet and the user types "=2+2", the user expects this to be entered in the formula bar; he will not want to have to click on the edit bar every time he wants to enter a formula. The application must determine if a keypress is intended for the spreadsheet or the edit bar.

For this reason, the application should define a subclass of **GenViewClass** and have the spreadsheet displayed in one of these objects. The subclass needs only one new thing: a handler for MSG_META_KBD_CHAR. When the view receives this message, it should decide whether to transfer the focus to the edit bar. If the character is a keyboard accelerator, a navigational key, or a key release, for example, the method should simply call its superclass method. However, if the keypress is intended for the edit bar (i.e. if it is an alphanumeric or punctuation character) the handler should send MSG_SSEBC_INITIAL_KEYPRESS to the edit bar. This message takes the same arguments as MSG_META_KBD_CHAR, so the MSG_META_KBD_CHAR handler can simply pass its own arguments along. When the edit bar receives this message, it will take the focus away from the spreadsheet's view, and will process that keypress and all succeeding keypresses until a navigational key is pressed. At that time, it will return the focus to the spreadsheet's view.

■ MSG_SSEBC_INITIAL_KEYPRESS

```
void     MSG_SSEBC_INITIAL_KEYPRESS(
         word   character,
         word   flags,
         word   state);
```

Spreadsheet objects are put in a subclass of **GenViewClass**. When this subclassed View receives a keypress (via MSG_META_KBD_CHAR), it must decide whether to process the keypress itself (by calling the superclass' handler for MSG_META_KBD_CHAR) or hand the keypress and the focus over to the spreadsheet edit bar. This message gives the keypress and the focus to the edit bar.

The arguments are the same as those to MSG_META_KBD_CHAR. Thus, the View can simply pass along the arguments it received with MSG_META_KBD_CHAR.

Source: Subclass of **GenViewClass** whose content is Spreadsheet object; sent by handler for MSG_META_KBD_CHAR.

Destination: SSEditBarControl object.

Parameters:

character	Key which was pressed. This is the *character* argument to MSG_META_KBD_CHAR.
flags	Character flags and shift state. This is the *flags* argument to MSG_META_KBD_CHAR.
state	ToggleState and scan code. This is the *state* argument to MSG_META_KBD_CHAR.

Return: Nothing.

Interception: This message is not ordinarily subclassed.

20.4 Other Spreadsheet Controllers

The Spreadsheet Library provides several controllers which are designed to work with the Spreadsheet objects. These controllers act transparently to the application; you need merely declare them in your source code, and the user will be able to use them to modify the spreadsheet at will. All of these controllers should be on the application's GAGCNLT_SELF_LOAD_OPTIONS GCN list.

Other controllers are continually being added; as new ones are written, applications will be able to incorporate them easily into existing spreadsheets.

20.4.1 The SSEditControl

This object implements the UI for clearing, inserting, and deleting cells.

20.4.2 Notes and SSNoteControlClass

Every cell in a spreadsheet actually contains two pieces of information. One of these is the cell's number, string, or formula. It is this which is displayed in the spreadsheet window and in the formula bar; when we speak of a cell's value, we mean the cell's number or string, or the value to which the formula evaluates.

However, every cell can have another piece of information, namely a *note*. The note is a null-terminated text string. The note does not have any effect on the cell's value; it is used entirely for commenting.

If you declare an SSNoteControl object, the user will be able to see and change the notes. The SSNoteControl is much like the formula bar; it automatically displays the note for the active cell, and the user will be able to change it at will.

20.4.3 Row and Column Size

Most spreadsheets will want to use the SSRowHeightControl and SSColumnWidthControl objects. These objects allow the user to change the height and width of the cells. As with the other controllers, these act transparently to the application.

20.4.4 Sorting and SSSortControlClass

A controller has been provided which allows users to sort a group of rows or columns. The user can specify whether the sort is done in ascending or descending order. If you are sorting rows, the column containing the active cell is the index column; it contains the values by which the rows will be sorted. (For example, if the active cell is B4 and the user sorts "ascending by row," the row with the lowest value in column 4 will be first.)

The Sort controller automatically uses the localization libraries to sort strings in the local language's alphabetical order.

20.4.5 Defining and Using Names

Users may often find it convenient to use names for cells or values. For example, a user may want to define "YearTotal" as a synonym for cell A1; whenever he uses YearTotal in an equation, the spreadsheet would use the value in cell A1. The Spreadsheet object supports this with the Name controller. The user can use the SSDefineNameControl controller to associate a name with a string, number, cell reference, or cell range. When the user enters a formula in a cell, the name will be passed through to the Parse routine. The evaluator will call the spreadsheet to find out what the current value of the name is and pass it back to the evaluator. All of this happens transparently to the application.

The user can also select one of the names he has defined, and paste it into the formula bar, by using the SSChooseNameControl object.

20.4.6 Headers and Footers

The user can define a range of cells as a "header". When the spreadsheet is printed, the header will be displayed at the top of every page. Similarly, the user can specify a "footer" which will appear at the bottom of every printed page. The user uses the SSHeaderFooterControl to specify these ranges.

20.4

◆Objects

Pen Object Library

21

The Pen Library provides routines and object classes which work together to form the backbone of a note-book style database storing pen input.

An Ink object is a visual object and may be used by any application that wants to work with pen input. Though any targetable object may accept ink (see “Input,” Chapter 11 of the Concepts Book), **InkClass** has many optimizations for working with ink.

The Ink Database routines provide a front end to the standard GEOS Database (DB) library routines well suited for storing and organizing several small pieces of information. These routines allow the storage of notes within a hierarchical arrangement of folders. Each note may contain one or more pages of textual or ink information.

If you wish to work directly with the incoming pen input, read “Input,” Chapter 11 of the Concepts Book to find out how to intercept pen events. To understand the inner workings of the Ink Database routines, you should be familiar with the DB library.

21.1 The Ink Object

InkClass provides methods for storing multiple pen inputs in a compact form. It automatically handles all queries about pen input. It handles display, with the power to display the ink in any color, and it allows the use of standard or custom background pictures.

To change the way the Ink class (or any other appropriate class) handles ink, the messages to subclass are MSG_META_NOTIFY_WITH_DATA_BLOCK (with the notification type MANUFACTURER_ID_GEOWORKS, NT_INK), MSG_META_QUERY_IF_PRESS_IS_INK, and MSG_VIS_QUERY_IF_OBJECT_HANDLES_INK.

21.1.1 Instance Data and Messages

When setting up an Ink object, probably the only pieces of instance data the application will be concerned with will be *II_flags*, *II_dirtyOutput*, and *II_dirtyMsg*.

Most of the flags are easy to understand, with the possible exceptions of the IF_HAS_TARGET and IF_DIRTY fields, which should not be set when creating the object in any case.

Code Display 21-1 InkClass Instance Data

```
/*      II_flags:
 *      This field holds flags governing the object’s behavior:
 *              IF_MOUSE_FLAGS,
 *              IF_SELECTING,
 *              IF_HAS_TARGET,
 *              IF_HAS_SYS_TARGET,
 *              IF_DIRTY,
 *              IF_ONLY_CHILD_OF_CONTENT,
 *              IF_CONTROLLED, (Set if to be used with an InkControl)
 *              IF_INVALIDATE_ERASURES,
 *              IF_HAS_UNDO */
@instance InkFlags              II_flags = IF_HAS_UNDO;

/*      II_tool:
 *      This field keeps track of how the user is interacting with the Ink. There
 *      are three possible tools: IT_PENCIL, IT_SELECTOR, and IT_ERASER. */
@instance InkTool       II_tool;

/*      II_penColor:
 *      The color to use when drawing ink. */
@instance Color         II_penColor = C_BLACK;

/*      II_segments:
 *      Do not set this field explicitly. This field is a handle to the chunk array
 *      containing the pen segments. The segments are stored as an array of Point
 *      structures. The stored coordinates are all positive; any x coordinate with
 *      its sign bit set belongs to the last point in a gesture. Thus, a small
 *      cross shape centered at (72, 72) might be stored:
 *              (0x0048, 0x0046)
 *              (0x8048, 0x004A) [note sign bit set in x coordinate]
 *              (0x0046, 0x0048)
 *              (0x804A, 0x0048) [sign bit set in x coordinate]*/
@instance ChunkHandle   II_segments;

/*      II_dirtyMsg, II_dirtyOutput:
 *      Together, these fields form an Action Descriptor. When the Ink processes
 *      a point of pen information, erases anything, or handles an undo event the
 *      IF_DIRTY flag will be set. If the flag was not set already, then the Ink
 *      will send the AD's message to the AD's object. The handler for this message
 *      should probably clear the IF_DIRTY bit. */
@instance optr          II_dirtyOutput;
@instance Message       II_dirtyMsg;

@instance Rectangle     II_selectBounds;        /* Internal */
@instance GStateHandle  II_cachedGState;        /* Internal */
@instance TimerHandle   II_antTimer;            /* Internal */
@instance word          II_antTimerID;          /* Internal */
@instance byte          II_antMask;             /* Internal */
```

Most of the Ink messages just change or retrieve the values of the instance fields. The exceptions are two messages will help those applications which need to save or transfer the Ink object's pen data. Use MSG_INK_SAVE_TO_DB_ITEM to save the pen data to an arbitrary DB item. If the application changes this information and wishes to pass it back to the ink object, use MSG_INK_LOAD_FROM_DB_ITEM.

■ MSG_INK_SET_TOOL

```
void    MSG_INK_SET_TOOL(
        InkTool tool);
```

This message allows the Ink to switch between pencil and eraser tools, changing the *II_tool* field.

Source: Unrestricted.

Destination: Any Ink object.

Parameters: *tool* A tool, either IT_PENCIL or IT_ERASER.

Return: Nothing.

Interception: Unlikely.

MSG_INK_GET_TOOL

```
InkTool  MSG_INK_GET_TOOL();
```

This message returns the Ink's present tool, as stored in *II_tool*.

Source: Unrestricted.

Destination: Any Ink object.

Parameters: None.

Return: The present tool, either IT_PENCIL or IT_ERASER.

Interception: Unlikely.

MSG_INK_SET_PEN_COLOR

```
void     MSG_INK_SET_PEN_COLOR(
         Color  clr);
```

This message changes the color used to draw the ink, changing the value in *II_penColor*.

Source: Unrestricted.

Destination: Any Ink object.

Parameters: *clr* Index to a palette (e.g. C_RED).

Return: Nothing.

Interception: Unlikely.

MSG_INK_SET_DIRTY_AD

```
void     MSG_INK_SET_DIRTY_AD(
         word    method,
         optr    object);
```

This message sets the Action Descriptor to be activated when the user dirties the object, changing the values in *II_dirtyMsg* and *II_dirtyOutput*.

Source: Unrestricted.

Destination: Any Ink object.

Parameters: *method* The message to send when the object is dirty.

object The object which should receive the above message.

Return: Nothing.

Interception: Unlikely.

◆Objects

■ MSG_INK_SET_FLAGS

```
void        MSG_INK_SET_FLAGS(
            InkFlags setFlags,
            InkFlags clearFlags);
```

This message changes the value of the *II_flags* field. Note that something which sets the IF_DIRTY bit should probably also perform the action stored in the *II_dirtyMsg* and *II_dirtyOutput* fields.

Source: Unrestricted.

Destination: Any Ink object.

Parameters: *setFlags* The flags to turn on.

clearFlags The flags to turn off.

Return: Nothing.

Interception: Unlikely.

■ MSG_INK_GET_FLAGS

```
InkFlags    MSG_INK_GET_FLAGS();
```

This message gets the value of the *II_flags* field.

Source: Unrestricted.

Destination: Any Ink object.

Parameters: None.

Return: The present value of the *II_flags* field.

Interception: Unlikely.

■ MSG_INK_SAVE_TO_DB_ITEM

```
void        MSG_INK_SAVE_TO_DB_ITEM(
            DBReturn            * RetValue,
            InkDBFrame          * ptr);
```

This message saves the Ink's pen data into the passed DB item. The pen data will be stored compressed. Calling this message sets the object not dirty.

Source: Unrestricted.

Destination: Any Ink object.

Parameters: *ptr* A pointer to an **InkDBFrame** structure, shown below.

RetValue A pointer to an empty **DBReturn** structure, to be filled by the handler.

Return: The structure pointed to by *RetValue* will contain the returned information.

Structures: The **InkDBFrame** and **DBReturn** structures are defined below:

```
typedef struct {
      Rectangle         IDBF_bounds;
            /* The bounds of the Ink data */
      VMFileHandle      IDBF_vmFile;
            /* VM file to write to*/
      DBGroupAndItem    IDBF_DBGroupAndItem;
            /* DB Item to save to
             * (or NULL to create a new one) */
      word              IDBF_DBExtra;
            /* Extra space to skip at start
             * of block */
} InkDBFrame;
typedef struct {
      word        DBR_group;
      word        DBR_item;
      word        DBR_unused1;
      word        DBR_unused2;
} DBReturn;
```

Interception: Unlikely.

■ MSG_INK_LOAD_FROM_DB_ITEM

```
void    MSG_INK_LOAD_FROM_DB_ITEM(
        InkDBFrame *ptr);
```

This message loads the compressed data into the Ink from the passed DB item. If a NULL handle is passed, then the Ink is cleared. This message marks the Ink as clean.

Source: Unrestricted.

Destination: Any Ink object.

Parameters: *ptr* A pointer to an **InkDBFrame** structure.

Return: Nothing.

Objects

Structures: For the **InkDBFrame** structure, see MSG_INK_SAVE_TO_DB_ITEM.

Interception: Unlikely.

21.1.2 Storing Ink to DB Items

Pen information comes in as a MSG_META_NOTIFY_WITH_DATA_BLOCK of type NT_INK accompanied by an array containing the coordinates visited by the pen. The pen data keeps track of the coordinates of the pen input. Every time pen input comes in, the ink object notes the coordinates. The ink object is optimized to save space. For instance, the Ink object eliminates collinear points: if three pen events are collinear, it will not record the middle one, recognizing it as redundant.

The non-redundant points are written out to the *II_segments* field, a chunk array of Point structures. Note that the coordinates are unsigned. If a point's *x* coordinate's sign bit is set, that does not mean that the *x* coordinate is negative; if this sign bit is set this is a signal that this point is the last point of a gesture.

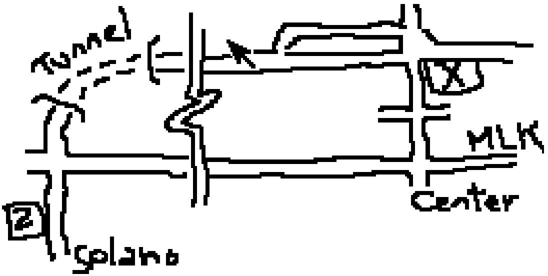


Figure 21-1 *Typical Pen Input*

Normally, pen input to an Ink object will be mostly made up of horizontal or vertical strokes.

When writing pen data to a DB item, the Ink object does some more compression. Applications which work with the items used by MSG_INK_SAVE_TO_DB_ITEM and MSG_INK_LOAD_FROM_DB_ITEM must work with this compression. Since the user is dragging the pen around in a continuous gesture, the pen events tend to occur close together. Thus, it is nice to have a way to record a coordinate as a small offset from another coordinate. Since many strokes are almost horizontal or vertical, quite often the horizontal or vertical offset will be zero or one.

To take advantage of these tendencies, the ink object stores pen input as a bitstream. Coordinates may be recorded either as absolute positions or as offsets from the last coordinate. See Table 21-1 for a list of components of this bitstream.

Table 21-1 *Components of the Ink's Bitstream*

21.1

Bit Pattern	Meaning	Total Bits
00	0 offset	2
01	+1 offset	2
10 00 000	terminate segment	7
10 00 001	+2 offset	7
10 00 010	+3 offset	7
10 00 011	+4 offset	7
10 00 100	+5 offset	7
10 00 101	+6 offset	7
10 00 110	+7 offset	7
10 00 111	+8 offset	7
10 01 000	(reserved for future use)	7
10 01 001	-2 offset	7
10 01 010	-3 offset	7
10 01 011	-4 offset	7
10 01 100	-5 offset	7
10 01 101	-6 offset	7
10 01 110	-7 offset	7
10 01 111	-8 offset	7
10 10 xxxxxx	6-bit keyword (reserved)	10
10 11 xxxx xxxx xxxx xxx	15-bit (unsigned) absolute position	19
11	-1 offset	2

When writing out a gesture to a DB item, the first point will always be recorded as an absolute position. Thus, first the *x* coordinate will be recorded, then the *y* coordinate. Each coordinate will be marked as absolute by the 1011 bit pattern.

For each subsequent pen point, the algorithm will first make sure that the new point is not collinear with the previous two. If it is, then the algorithm will make the incoming pen event overwrite the previous event's coordinates.

Objects

For each event, the algorithm will first write out the *x* coordinate, then the *y* coordinate.

- If the coordinate is at 0 or 1 offset from the previous coordinate, the algorithm will write out the appropriate two-bit code (00, 01, or 11).
- If the coordinate is at an offset from the previous coordinate between 2 and 8, then the algorithm will write out the appropriate 7 bit code (1000xxx for a positive offset, 1001xxx for a negative offset).
- If the coordinate is more than 8 points from the previous coordinate, the algorithm writes out 1011 followed by the absolute coordinate, represented as a 15 bit unsigned quantity.

When the input is finished, the algorithm writes a 1000000 bit pattern, signalling the end of the segment.

Decompressing the data is a matter of traversing the bitstream and detecting the appropriate patterns.

As an example of how the algorithm compresses pen input, suppose the Ink object were writing the following gesture to a DB item:

```
(72, 71)
(82, 74)
(84, 74)
(85, 72)
```

The first coordinate is 72, so the algorithm will write out:
1011 (signals absolute coordinate) *000000001001000*
The second coordinate is 71, so after handling the second coordinate, the stream will be:
1011 000000001001000 *1011 000000001000111*
The *x* coordinate of the second point is 82, which is 10 points away from the previous *x* coordinate. Unfortunately, this is too far to express as a short offset, so the algorithm writes another absolute coordinate (the new part of the stream is shown in italics):
1011 000000001001000 1011 000000001000111 *1011 000000001010010*
The *y* coordinate of the second point is 74, at a positive 3 offset from the previous *y* value, so the algorithm will write out the appropriate offset code instead of an absolute position code:
…1011 000000001000111 1011 000000001010010 *10 00 010*
The third point’s *x* coordinate is 84, at a +2 offset from 82. The *y* coordinate

is 74, the same as the previous point's *y* coordinate:

…1011 000000001000111 1011 000000001010010 10 00 010 *10 00 001 00*

The last point's *x* coordinate is one higher than the previous; its *y* coordinate is two less.

…1011 000000001010010 10 00 010 10 00 001 00 *01 10 01 001*

Since it has reached the end of the pen input (this was a suspiciously short gesture, a somewhat contrived example), the algorithm then writes an end-of-segment code:

…1011 000000001010010 10 00 010 10 00 001 00 01 10 01 001 *10 00 000*

If the Ink object were holding more than one gesture of information, it would write the next gesture's elements starting after the end-of-segment code of the first.

21.2 Working with the Ink DB

The Ink Database provides a simplified, specialized API to the database library. It allows the user to organize pieces of information on notes stored in a hierarchy of folders. Each note may have one or more pages, with each page corresponding to the contents of an ink or text object. The data stored in each page is a DB item returned by MSG_INK_SAVE_TO_DB_ITEM (for Ink objects) or MSG_VIS_TEXT_GET_ALL_DB_ITEM (for Text objects).

Notes and folders are specified by means of a dword identifier. This identifier has nothing to do with where the note's (or folder's) data is stored, or where it appears in the folder tree. Applications should use the **InkLoadPage()** and **InkSavePage()** routines to work with a note's data, and use the routines described below to determine where a note or folder appears in the folder tree.

21.2.1 Getting Started

`InkDBInit()`

To create an Ink Database, an application needs a file handle, perhaps the file holding a GenDocument's data. Before calling any other Ink Database functions, call **InkDBInit()** to set up the file correctly; this routine should be called exactly once per Ink DB. If the database is part of a GenDocument, then this routine should be called within the MSG_GEN_DOCUMENT_INITIALIZE_DOCUMENT_FILE. Other routines (described below) which might be appropriate when first setting up an Ink Database include **InkSetDocPageInfo()** and **InkSetDocGString()**.

21.2.2 Displaying the Data

```
InkNoteLoadPage(), InkNoteSavePage(), InkGetDocPageInfo(),
InkSetDocPageInfo(), InkSetDocGString(),
InkGetDocGString(), InkNoteGetNoteType()
```

Assuming that the application is using text and ink objects to display the information held in the Ink DB, use **InkNoteLoadPage()** and **InkNoteSavePage()** to transfer information between the Ink object and the Ink DB. **InkNoteLoadPage()** loads an ink or text object with the data stored within the passed note. Use **InkNoteGetNoteType()** to determine what sort of data is stored within the note. Once the user has made changes, those changes should be stored to the database. Call **InkNoteSavePage()** to write the changes.

To find out the document size associated with an Ink Database, call **InkGetDocPageInfo()**. To change the page size, call **InkSetDocPageInfo()**.

The Ink DB routines support the notion of a background picture for ink information. There is one background picture for the entire database. To set the background picture, use **InkSetDocGString()**. To find out the current background picture, call **InkGetDocGString()**.

The background GString is stored in VM; call **GrLoadGString()** and **GrDrawGString()** to draw it.

21.2.3 Titles and Keywords

```
InkNoteSetKeywords(), InkNoteSetKeywordsFromTextObject(),
InkNoteGetKeywords(), InkNoteSendKeywordsToTextObject(),
InkGetTitle(), InkSendTitleToTextObject(),
InkFolderSetTitle(), InkFolderSetTitleFromTextObject(),
InkNoteSetTitle(), InkNoteSetTitleFromTextObject()
```

Each note may have two text strings which are helpful for identification: a title and a set of keywords. These words may be used as the fields for a computed search if the application supports these; regardless, the user will certainly find these fields useful for organizing notes.

21.2

To set a note's title, call **InkNoteSetTitle()**. There is a corresponding **InkFolderSetTitle()** for setting the title of a folder. Since applications may wish to set the titles of these items based upon the user's entry in a text object, there are two routines **InkNoteSetTitleFromTextObject()** and **InkFolderSetTitleFromTextObject()** which take an item's name from a text object. **InkGetTitle()** gets any item's title, and **InkSendTitleToTextObject()** is a specialized function used to update the passed text object's text to hold the item's title. The maximum length of any title should be INK_DB_MAX_TITLE_SIZE.

Notes may have keywords: words which should not appear in the title but which are still useful for searches. Folders do not have keywords. To set a note's keywords, use **InkNoteSetKeywords()**; to use the contents of a text object as the keywords, use **InkNoteSetKeywordsFromTextObject()**. To retrieve the keywords, call **InkNoteGetKeywords()**. **InkNoteSendKeywordsToTextObject()** replaces a text object's text with the passed note's keywords. The maximum length of any keyword should be INK_DB_MAX_NOTE_KEYWORDS_SIZE.

21.2.4 Navigating the Folder Tree

```
InkDBGetDisplayInfo(), InkDBSetDisplayInfo(),
InkDBGetHeadFolder(), InkGetParentFolder(),
InkFolderGetContents(), InkFolderGetNumChildren(),
InkFolderDisplayChildInList(), InkFolderGetChildInfo(),
InkFolderGetChildNumber(), InkNoteGetNumPages()
```

Assuming the application allows the existence of more than one folder, it must allow some way to move around within the folder tree. If the application allows the user to change the structure of the folder tree, then it will need UI which allows the user to navigate an arbitrary tree. There are routines to find out and change which page is being displayed. For those applications which will need to get information about the folder tree, there are routines to get information about the folder tree.

To find the application's current location within the DB, call **InkDBGetDisplayInfo()**. This routine returns the current folder ID, the note ID if any is selected, and the page number within the note. To go to a different location, call **InkDBSetDisplayInfo()**. To use this routine, the application must pass a folder ID, along with a valid note ID and page number if a note is to be selected.

Chances are the user will be maneuvering within the folder tree. To get the ID of the root folder, use **InkDBGetHeadFolder()**. To find the parent folder of the passed parent or note, call **InkGetParentFolder()**. **InkFolderGetContents()** returns two chunk arrays, one containing the double word identifiers of all the folder's subfolders, the other containing the identifiers of the folder's child notes. **InkFolderGetNumChildren()** returns the number of subfolders and notes within a folder.

To display a note or folder's name in a GenDynamicList, use **InkFolderDisplayChildInList()**. This routine comes in handy when constructing UI for navigating the folder tree. To copy the icon and folder or note name of a folder or note into the visual moniker of an entry in a list, call **InkNoteCopyMoniker()**.

To get information about a folder's child, call **InkFolderGetChildInfo()**. This routine returns a bit specifying whether the child is a folder or note, along with the child's ID number. The **InkFolderGetChildNumber()** routine returns the passed child's place number within the folder.

21.2.5 Managing Notes and Folders

```
InkFolderCreateSubFolder(), InkFolderMove(),
InkFolderDelete(), InkNoteCreate(), InkNoteDelete(),
InkNoteMove(), InkNoteCreatePage()
```

Some Ink DB applications might just create a hierarchy of notes and not allow the user to move or change notes. Applications that will move notes and folders should use the following functions to make changes.

The **InkFolderCreateSubFolder()** routine creates a new folder as a child of the passed existing folder. Use **InkFolderMove()** to move a folder to a new parent folder. **InkNoteMove()** similarly moves a note to a new parent folder. **InkFolderDelete()** deletes a folder and all subfolders and notes that folder contained. **InkNoteCreate()** creates a new note. **InkNoteDelete()** deletes a note. **InkNoteCreatePage()** adds a new page to a note.

21.2

21.2.6 Manipulating Notes

```
InkNoteGetPages(), InkNoteGetNumPages(),
InkNoteSetModificationDate(),
InkNoteGetModificationDate(), InkNoteGetCreationDate(),
InkNoteSetNoteType(), InkNoteGetNoteType()
```

Normally, the note will store information supplied by an Ink or Text object. However, applications may work with a note's information directly. Call **InkNoteGetPages()** to get the DB item in which the note's information is stored. The DB item contains a chunk array; each entry of the array contains the information for one page (the DB item associated with an Ink or Text object). To find out how many pages there are in a given note, call **InkNoteGetNumPages()**.

The note will be expecting either text or ink; call **InkNoteSetNoteType()** to specify what sort of data will be coming in. The note type is specified by means of a **NoteType** value: NT_INK or NT_TEXT. To find out a note's type, call **InkNoteGetNoteType()**.

When writing changes, you may wish to update the note's modification date. Call **InkNoteSetModificationDate()** to update this information. To find out the date last modified, call **InkNoteGetModificationDate()**. To find out the date the note was created, call **InkNoteGetCreationDate()**.

21.2.7 Searching and Traversing the Tree

`InkNoteFindByTitle(), InkNoteFindByKeywords(), InkFolderDepthFirstTraverse()`

Sometimes the user will remember what a note is called, but has lost it in the tree of folders. Sometimes the user will want to find all notes which contain a certain keyword. Use **InkNoteFindNoteByTitle()** to get a buffer containing IDs of all notes whose titles match the passed string. **InkNoteFindNoteByKeywords()** similarly returns a buffer containing the IDs of all notes with matching keywords.

For more complicated commands, **InkFolderDepthFirstTraverse()** allows the application to perform a depth-first traversal of the folder tree, calling the passed routine with all encountered folders.

21.3 InkControlClass

InkControlClass, a subclass of **GenControlClass**, provides a menu which allows the user to select an Ink tool for use with an Ink object.

Code Display 21-2 InkControlClass Features

```
typedef ByteFlags InkControlFeatures;
/* These features may be combined using | and &:
        ICF_PENCIL_TOOL,
        ICF_ERASER_TOOL
        ICF_SELECTION_TOOL */

typedef ByteFlags InkControlToolboxFeatures;
/* These features may be combined using | and &:
        ICTF_PENCIL_TOOL,
        ICTF_ERASER_TOOL
        ICTF_SELECTION_TOOL */

#define IC_DEFAULT_FEATURES             (ICF_PENCIL_TOOL | ICF_ERASER_TOOL | \
                                         ICF_SELECTION_TOOL)
#define IC_DEFAULT_TOOLBOX_FEATURES     (ICTF_PENCIL_TOOL | ICTF_ERASER_TOOL | \
                                         ICTF_SELECTION_TOOL)
```

```
/* Add this controller to the application’s self-load options GCN list. */
```

21.3

◆Objects

Config Library

22

Not all users are the same; not all computing environments are the same. Often some part of an application allows the user to tailor the UI to his wishes. The user may wish to specify some preferences at a system level rather than on an application level. To do this, the user will look to the Preferences Manager application. The **config** library provides mechanisms for writing Preferences modules.

Normally the purpose of a Preferences module's gadgetry is to set up some information in the user's .INI file. If you are encouraging other developers to take advantage of the configuration done by your module, be sure to document the .INI file categories and keys affected.

There are three main tasks involved in writing a Preferences module:

- Providing a tree of UI gadgetry which will act as the module's dialog box.
- Providing information Preferences will use to construct the module's button.
- Providing any underlying functionality other than writing information to the .INI file.

The first two routines exported in the module's .gp file will be responsible for carrying out the first two of these tasks. The final task is difficult to define—it will depend on the nature of your module.

Preferences modules written in C will need one more thing. To indicate that the preferences module is compatible with the standard GEOS 2.0 Preferences Manager applications, the module will need a local.mk makefile for compilation, and that local.mk file should include a line that reads:

```
	_PROTO = 2.0
```

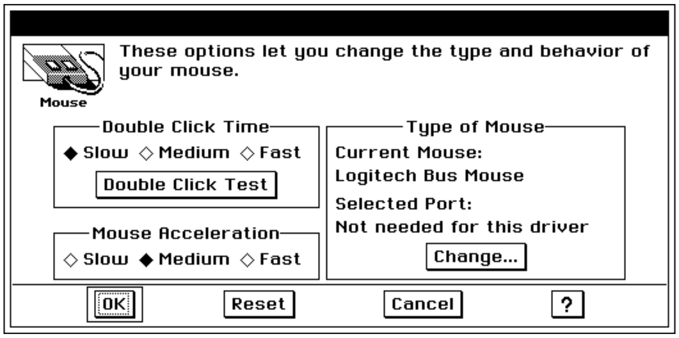
For information about local.mk files, see "Using Tools," Chapter 10 of the Tools book.

Before you try to write a Preference module, there are some things you should already know. Each module is a library, so be sure to read "Libraries," Chapter C of the Concepts Book to find out how to write a GEOS library. To use the information your module will write to the .INI file, read the Initialization File section of "Applications and Geodes," Chapter 6 of the Concepts Book.

22.1 Providing the UI

There are two steps to providing the UI for your preferences module. The first is to declare the objects you will use. The second is to write and export a routine which returns the head of the tree of UI gadgetry.

22.1



These options let you change the type and behavior of your mouse.

Double Click Time
◆ Slow ◇ Medium ◇ Fast

Mouse Acceleration
◇ Slow ◆ Medium ◇ Fast

Type of Mouse
Current Mouse:
Logitech Bus Mouse
Selected Port:
Not needed for this driver

Figure 22-1 *Preferences Dialog*

22.1.1 Designing the UI Tree.

Preferences modules present a dialog box to the user. You will need to design the dialog for your module. As you do so, you may find objects of the following classes useful. Each of the following classes is based upon one of the Generic UI classes.

GenClass provides mechanisms by which objects will read and write values to the .INI file. Pref- classes provide further functionality, such as storing an "original" value to revert to if the user decides to Cancel their changes.

The following Pref- classes are available:

PrefDialogClass

This class specializes in acting as the root of a Pref module's UI tree. It has been set up to provide "OK" and "Cancel" triggers by default. It has a mechanism to restart the system on an Apply if the module specifies that such a reset will be necessary to implement the user's changes.
Often a module will use a subclass of **PrefDialogClass** as the head of its UI tree—often the subclass intercepts

MSG_GEN_APPLY to ensure valid user choices and to take other appropriate actions.

PrefValueClass
PrefValue objects act like objects of **GenValueClass**. However, these objects have the ability to reset their values to that originally stored in the .INI file (in case the user wants to cancel). They can handle only word-length values.

PrefItemGroupClass
Objects of this class act like GenItemGroups. In addition, this class has the ability to store an "original" value.

PrefStringItemClass
A PrefItemGroup normally writes data about selected items in the form of an integer. However, if its children are PrefStringItem objects, the PrefItemGroup can instead write out a special string associated with each PrefStringItem, resulting in a more readable .INI file.

PrefBooleanGroupClass
This class acts like **GenBooleanGroupClass** with the ability to store an "original" value.

PrefDynamicListClass
Use objects of this class where you might otherwise use a GenDynamicList. This class will not work together with PrefStringItem objects.

PrefTocListClass
This class presents an alphabetized list of files or driver names. It is used to provide lists of devices, such as the list of available printers. It can also be used to show a list of files with a given token in a given directory.

TitledGlyphClass
This specialized subclass of GenGlyph shows both an icon and a text moniker. It is used to present the icon and name at the top of a typical Preferences module dialog box.

PrefInteractionClass
This class acts like **GenInteractionClass**, but also will relay certain messages to its children which GenInteraction would not. Those objects which will need to work with the .INI file should receive these messages, so any Gen- objects which will

work with the .INI file and any Pref objects should be grouped under PrefInteractions instead of GenInteractions.

PrefTextClass
This class behaves like **GenTextClass**, except that it will load and save its value based upon that stored in the .INI file.

PrefTriggerClass
This class acts as does GenTriggerClass, but has an extra action, so that two separate messages will be sent when the trigger is activated.

PrefControlClass
This class acts as a cross between **PrefClass** and **GenControlClass**.

PrefTimeDateControlClass
This class allows the user to set the system date and time.

PrefClass **PrefClass** is something like **GenClass**—while never used directly, it is the superclass of all the other Pref- classes, and sets up several of the mechanisms which all will use.

There aren't any special restrictions on what sorts of objects appear in the UI tree of a Preferences module. However, the following rules will prove useful in constructing modules that correctly write their data to the .INI file and have a look and feel consistent with existing modules. For an example, see Code Display 22-1.

- As a rule of thumb, the top object of the tree should be a PrefDialog.
- For a consistent look, the upper portion of the dialog should contain the module's icon, name, and some brief help text. This is normally done using a TitledGlyph and a GenText object.
- Generally the easiest way to specify the .INI file category and key to use is by means of ATTR_GEN_INIT_FILE_CATEGORY and ATTR_GEN_INIT_FILE_KEY. Assuming that the module will work with just one category, it is sufficient to declare an ATTR_GEN_INIT_FILE field for just the top object of the module's UI tree.
- Because of the way messages are relayed, you should use PrefInteractions instead of GenInteractions if any objects under the interaction will be saving data to the .INI file.

Objects

Code Display 22-1 Pref Module UI Framework

```
@object MPMDialogClass MPMRoot = {
        GI_states = @default & ~GS_USABLE;
        GI_comp = @MPMTitleGroup, @MPMOtherStuff;
        HINT_INTERACTION_SINGLE_USAGE;
        HINT_INTERACTION_COMPLEX_PROPERTIES;
        HINT_ORIENT_CHILDREN_VERTICALLY;
        HINT_LEFT_JUSTIFY_CHILDREN;
        ATTR_GEN_HELP_CONTEXT = “myPrefModule”;
        ATTR_GEN_INIT_FILE_CATEGORY = “myPref”;
}

@object GenInteractionClass MPMTitleGroup = {
        GI_comp = @MPMTitle, @MPMHelp;
        HINT_ORIENT_CHILDREN_HORIZONTALLY;
        HINT_EXPAND_WIDTH_TO_FIT_PARENT;
}

@object TitledGlyphClass MPMTitle = {
        GI_visMoniker = list {
                @FontTextMoniker, @FontLCMoniker, @FontLMMoniker, @FontLCGAMoniker
        }
}

@object GenTextClass MPMHelp = {
        GI_attrs = @default | GA_READ_ONLY;
        GTXI_text = “Do such and such to configure your so and so.“;
        HINT_EXPAND_WIDTH_TO_FIT_PARENT;
        /* Might want HINT_MINIMUM_SIZE */
}
```

22.1.2 UI Fetch Routine

Once you have constructed the UI tree, write a routine that returns the head of that tree. This should be the first routine exported in your module’s .gp file. The Preferences Manager knows to call the first exported routine when it needs the UI tree for a module.

The routine should have the parameters

```
optr _pascal (void)
```

For an example of such a routine, see Code Display 22-2.

Code Display 22-2 UI Fetch Routine

```
optr _pascal MPMGetPrefUITree(void) { return (@MPMRoot);}
```

22.2

22.2 Module Information Routine

The Preferences application needs to know some information about each module to present the trigger which allows the user into said module. Preferences does this by calling the second routine exported by the module library. Preferences expects this routine to fill in a blank **PrefModuleInfo** structure.

By filling in this structure, the routine describes the button that Preferences will use to represent the module. Also, you may specify a User Level for the module, so it will be presented only to users with enough expertise to use it. You may also ask that only users with certain privileges be given access to the module; without the proper privileges, they won't see it.

The Routine should have the parameters

```
void _pascal (*PrefModuleInfo)
```

For an example of such a routine, see Code Display 22-3.

Code Display 22-3 Module Information Routine

```
void _pascal MPMGetModuleInfo(PrefModuleInfo *info)

{       /* We'll set up this structure for both system and user information */
        moduleInfo->PMI_requiredFeatures = 0;
        moduleInfo->PMI_prohibitedFeatures = 0;
        moduleInfo->PMI_minLevel = UIIL_ADVANCED;
        moduleInfo->PMI_maxLevel = UIIL_MAX_LEVEL;
        moduleInfo->PMI_monikerList = @MPMMonikerList;
        moduleInfo->PMI_monikerToken = moduleToken;
}
```

Objects

```
const GeodeToken moduleToken = { “MyPf”, MANUFACTURER_ID_MINE };
```

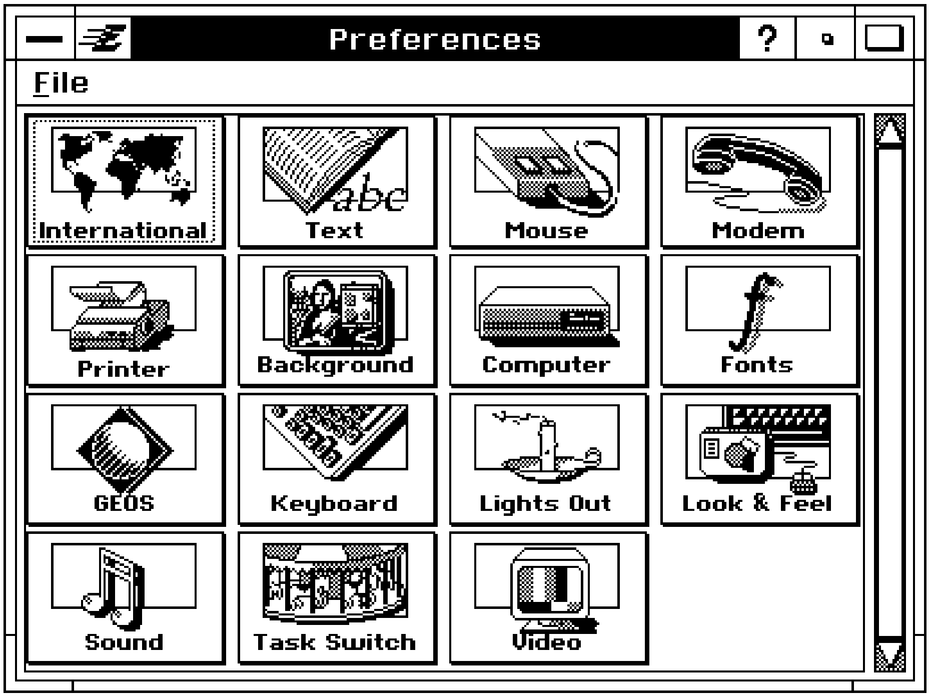


Figure 22-2 *Preferences Buttons*

The following information about the **PrefModuleInfo** structure may prove useful when writing the routine:

PMI_requiredFeatures
This field allows you to restrict the display of your module so that it will only appear to users which have certain privileges.

PMF_HARDWARE
These settings are for a user who has permissions to actually change the configuration of the workstation. In a network environment where users log in to different machines at different times, normal users would be prevented from changing the mouse drivers, video drivers, etc.

PMF_SYSTEM
These changes are more complex and potentially more damaging than the basic “user” changes, therefore, some users may be prevented from using these settings.

PMF_NETWORK
These are network settings. Generally only the system administrator should see these settings, as they affect the entire network.

22.2

PMF_USER These are basic user changes. These settings are the most basic and least dangerous, controlling user preferences such as background color and screen saver types.

PMI_prohibitedFeatures
This flag field allows you to restrict the display of your module so that it will only appear to users who *don’t* have certain privileges. If you had two modules, advanced UI and simple UI, one might be for the sysop and the other for normal users—this would keep the sysop’s Preferences from being “cluttered” by the plain user module. The flags available are the same as those for *PMI_requiredFeatures*.

PMI_minLevel
This field allows you to specify the minimum User UI Level in which your module should appear. If the module’s UI is very complicated and will confuse novice users, use this field to hide it from them.

PMI_maxLevel;
This field allows you to specify the *maximum* User UI Level in which your module should appear. If you have two versions of your module—one for advanced users and one for novice users—use this field to hide the novice module from the advanced users.

PMI_monikerList
Like the moniker list for an application, this will provide the icon shown on the main Preferences screen. Create this using the Icon editor. You should have icon formats 64x40 color, 64x40 monochrome, and 64x18 monochrome.

PMI_monikerToken
A four character token and manufacturer ID by which to recognize the module.

Objects

22.3 Important Messages

If you're going to create a subclass for use in a Preferences module, the following messages will be of special interest, as they signal that the user is carrying out certain important actions.

MSG_PREF_INIT
When a dialog box containing Pref objects is brought up, each object will receive MSG_PREF_INIT. Developers can subclass MSG_PREF_INIT to perform basic initialization procedures.

MSG_META_LOAD_OPTIONS
Generic objects (and by inheritance, Pref- objects) respond to MSG_META_LOAD_OPTIONS by loading a value from the appropriate place in the .INI file. This place is determined by the object's ATTR_GEN_INIT_FILE_KEY field. If the object has an ATTR_GEN_INIT_FILE_CATEGORY field, that category will be used; otherwise the default handler will look up the Generic tree for an object with a ATTR_GEN_INIT_FILE_CATEGORY field defined and will use the value stored there.
Unlike a GenInteraction, a PrefInteraction will relay this message to its children. When it's time to load options from the .INI file, a MSG_META_LOAD_OPTIONS will be sent to the top level of the module's UI tree. This message should be propagated down the tree to reach all objects which should read their values from the .INI file. This is why it is suggested that you use PrefInteractions in place of GenInteractions if any Pref- objects will be underneath. If you create a new sort of object which should relay this message to other objects, be sure to do so.

MSG_META_SAVE_OPTIONS
Generic objects (and by inheritance, Pref- objects) respond to MSG_META_SAVE_OPTIONS by saving a value to the appropriate place in the .INI file. This place is determined by the object's ATTR_GEN_INIT_FILE_KEY field. If the object has an ATTR_GEN_INIT_FILE_CATEGORY field, that category will be used; otherwise the default handler will look up the Generic tree for an object with a ATTR_GEN_INIT_FILE_CATEGORY field defined and will use the value stored there.
Unlike a GenInteraction, a PrefInteraction will relay this message to its children. When it's time to save options to the

.INI file, a MSG_META_SAVE_OPTIONS will be sent to the top level of the module's UI tree. This message should be propagated down the tree to reach all objects which should write their values to the .INI file. This is why it is suggested that you use PrefInteractions in place of GenInteractions if any Pref- objects will be underneath. If you create a new sort of object which should relay this message to other objects, be sure to do so.

Often, interceptors of this message will carry out whatever actions are necessary to carry out the user's wishes in those cases where writing something to the .INI file is not enough to bring the change about.

MSG_GEN_RESET

If the user clicks on a "Revert" or "Cancel" button, all values in UI gadgets should revert to those values originally stored in the .INI file. You will note that many Pref- objects have an extra field to store their "original" value. If you create your own kind of Preferences object, keep in mind that it will need some way to store the "original" value (probably initialized while handling MSG_META_LOAD_OPTIONS) and should intercept MSG_GEN_RESET to replace its value with the "original" value.

MSG_GEN_LOAD_OPTIONS

Generic objects (and thus Pref- objects) respond to MSG_META_LOAD_OPTIONS by sending themselves this message. This message is only meant to be used for managing the .INI file; any other functionality should occur in the handling of MSG_META_LOAD_OPTIONS.

MSG_GEN_SAVE_OPTIONS

Generic objects (and thus Pref- objects) respond to MSG_META_SAVE_OPTIONS by sending themselves this message. This message is only meant to be used for managing the .INI file; any other functionality should occur in the handling of MSG_META_SAVE_OPTIONS.

If you are interested in detecting certain user actions, the following list may prove useful:

User Opens Dialog

Any objects which need to do some initialization when the user first opens the module's dialog box should intercept

MSG_PREF_INIT. After receiving MSG_PREF_INIT, each object will receive a MSG_META_LOAD_OPTIONS.

User clicks Apply
Assuming the top object of the module's UI tree is a PrefDialog and that the dialog has been set up as a Properties dialog (as it is by default), then this object will receive a MSG_GEN_APPLY when the user activates the Apply trigger. The Pref dialog responds to this message by sending itself a MSG_META_SAVE_OPTIONS, which it will then pass on to its children.

User clicks Cancel or Reset
All Gen- and Pref- objects within the UI tree will receive a MSG_GEN_RESET.

22.4 Object Class Reference

If you will use any of the Pref- classes in your module, the following reference material may prove useful.

22.4.1 PrefClass

PrefClass defines several fields of instance data which will be used by all of its subclasses.

There are several variable data fields which make sure that a given Preference gadget will not be usable by those without the proper privileges; other fields make sure that the gadgets will not appear to those users whose UI levels indicate that they would be confused by the presence of such an object. Keep in mind that if all gadgets in your module demand some privilege or User level, you should reflect this in the routine which provide Preferences module information about your module.

Code Display 22-4 PrefClass Instance Data

```
@class PrefClass, GenClass, master, variant;
```

```
@instance PrefAttributes PI_attrs = (PA_LOAD_IF_USABLE|PA_SAVE_IF_USABLE);

typedef ByteFlags PrefAttributes;
#define PA_REBOOT_IF_CHANGED 0x80
/* This bit signals that changes in the state of
 * this object requires a system reboot to take effect. */

#define PA_LOAD_IF_USABLE 0x40
 /* Load options only if this object is usable (this is ON by default). */

#define PA_SAVE_IF_USABLE 0x20
 /* Save options only if this object is usable (this is ON by default) */

#define PA_SAVE_IF_ENABLED 0x10
 /* Save options only if this object is enabled */

#define PA_SAVE_IF_CHANGED 0x08
 /* Save options only if this object has changed. */

@vardata optr ATTR_PREF_REBOOT_STRING;
/* This is the string passed to MSG_PREF_MGR_DO_REBOOT_DIALOG -- the
 * string must be in a SHARABLE LMEM resource and is incorporated into the
 * middle of another sentence. In English, the sentence is “The system software
 * will shut down and restart to change the <reboot_string>. Do you wish
 * to proceed?”
 *
 * If an object has its PA_REBOOT_IF_CHANGED attribute set, it will scan for
 * this attribute, first in itself, then up the generic tree, until it
 * finds one. */
@reloc ATTR_PREF_REBOOT_STRING, 0, optr;

@vardata PrefMgrFeatures ATTR_PREF_REQUIRED_FEATURES;
/* features bits that must be set for this object to be usable. */

@vardata PrefMgrFeatures ATTR_PREF_PROHIBITED_FEATURES;
/* features bits that must not be set for this object to be usable. */

@vardata UIInterfaceLevel ATTR_PREF_MIN_LEVEL;
/* interface level below which this object will not be usable */

@vardata UIInterfaceLevel ATTR_PREF_MAX_LEVEL;
/* interface level above which this object will not be usable */
```

Objects

In addition to the instance data fields, there are messages which allow for the dynamic retrieval and updating of these fields.

■ MSG_PREF_SET_INIT_FILE_CATEGORY

```
void MSG_PREF_SET_INIT_FILE_CATEGORY(
        const char *category);
```

Pref objects automatically respond to this message by changing the value of the ATTR_GEN_INIT_FILE_CATEGORY variable data field. This is a utility message; similar functionality is available via MSG_META_ADD_VAR_DATA.

Source: Anywhere

Destination: Any **PrefClass** object.

Parameters: *category* Null-terminated category string.

Return: Nothing.

■ MSG_PREF_HAS_STATE_CHANGED

```
Boolean MSG_PREF_HAS_STATE_CHANGED();
```

This message determines whether the object or any of its children have changed since the last time options were loaded. This is used in conjunction with the PA_REBOOT_IF_CHANGED flag that can be set for an object—this flag signals that, if the object changes state, then GEOS should be restarted for the change to go into effect.

Source: **PrefClass** object sends this message to itself when handling MSG_PREF_GET_REBOOT_INFO.

Destination: Self.

Parameters: None.

Return: Should return *true* (i.e. non-zero) if state has changed, *false* (i.e. zero) otherwise.

Interception: New **PrefClass** subclasses will use this to compare the object's current state against the "original" state, assuming one was stored.

■ MSG_PREF_GET_REBOOT_INFO

```
optr MSG_PREF_GET_REBOOT_INFO();
```

This message asks whether the system will need to reboot as a result of changes. If a reboot is necessary, the dialog will then notify the user of the reboot, and give the user a chance to abort (since the system reset may be a lengthy process on some systems).

Source: PrefDialog sends this message to all children on an apply.

Destination: Any Pref- object.

Interception: Subclasses should check the PA_REBOOT_IF_CHANGED flag—if the flag is set, the object should compare its present value to its “original” value. If a reboot is never needed, then do nothing.

If a reboot is necessary, the method should return the string to use when rebooting. The standard way to do this is via MSG_PREF_GET_REBOOT_STRING.

If possible, developers should try to avoid situations where a system reset is necessary, and use mechanisms such as the General Change Notification instead.

Parameters: None.

Return: Zero if no reboot needed; otherwise the OD of string to insert in the confirm-shutdown dialog box. The string will be inserted into a sentence; the string should describe what is changing. You may wish to use the value of ATTR_PREF_REBOOT_STRING.

See Also: MSG_PREF_GET_REBOOT_STRING, ATTR_PREF_REBOOT_STRING.

■ MSG_PREF_GET_REBOOT_STRING

```
optr MSG_PREF_GET_REBOOT_STRING();
```

This message should return the optr of a string in a sharable local memory resource. This string will display in the dialog box which asks the user if it is all right to shut down. If a given object doesn’t know of a good string to supply, it should forward the message on to its parent in the generic tree. The default **PrefClass** handler returns the value stored in ATTR_PREF_REBOOT_STRING.

Source: Pref- object sends this message to itself.

Destination: Any **PrefClass** object.

Parameters: Nothing.

Return: Object Pointer of string (or NULL if not found).

■ MSG_PREF_INIT

```
void MSG_PREF_INIT(
         PrefMgrFeatures     features,
         UIInterfaceLevel    level);
```

This message initializes the object. The object should perform any required set-up to appear on-screen. The PrefDialog object will send this object to all of its children when the dialog is first initiated.

Source: Generic Parent

Destination: Any Pref- object.

Parameters: *features* This structure tells you a bit what sorts of changes the user prepared to make.

level The user's User Level.

Return: None.

Interception: Default behavior is to determine whether to make the object usable based on UI level and features (or will be, when implemented). Subclass should call superclass before handling, as subclass behavior may be different if the object is not usable,

■ MSG_PREF_NOTIFY_DIALOG_CHANGE

```
void MSG_PREF_NOTIFY_DIALOG_CHANGE(
         PrefDialogChangeType   type);
```

This message goes out via PDGCNLT_DIALOG_CHANGE list bound to PrefDialog object telling interested parties of a change in the box's state. It is sent when the dialog box is being opened, closed, or destroyed.

Source: PrefDialog object.

Destination: Any Pref- object registered on PDCGCNLT_DIALOG_CHANGE.

Interception: This message has no default handler; it must be intercepted to do anything useful.

Structures:

```
typedef enum {
      PDCT_OPEN,
      PDCT_CLOSE,
      PDCT_DESTROY,
      PDCT_RESTART,
      PDCT_SHUTDOWN
} PrefDialogChangeType;
```

■ MSG_PREF_SET_ORIGINAL_STATE

```
void MSG_PREF_SET_ORIGINAL_STATE();
```

This message copies the object's current state to the "original" value field, which will be used on a MSG_GEN_RESET.

Parameters: None.

Return: Nothing.

Interception: Subclasses should respond to this message by copying their current value to whatever instance field represents the "original" state of the object.

22.4.2 PrefValueClass

PrefValue objects behave like GenValue objects except that they have the ability to store an "original" value in the *PVI_originalValue* instance data field. Unlike GenValue objects, PrefValues can only handle single word values.

Code Display 22-5 PrefValueClass Instance Data

```
@class PrefValueClass, PrefClass;

@instance word PVI_originalValue = 0;

@vardata void ATTR_PREF_VALUE_ORIG_IS_INDETERMINATE;
```

```
@vardata word ATTR_PREF_VALUE_ROUND;
/* Specifies an integer value to which the value will be rounded on .INI file
 * reads/writes. For instance, with a value for 10 here, the value would always
 * round off to the nearest 10. Note that the user can still type in an illegal
 * value, but this value will be rounded before saving to the .INI file.*/
```

■ MSG_PREF_VALUE_SET_ORIGINAL_VALUE

```
void MSG_PREF_VALUE_SET_ORIGINAL_VALUE(
        word                orig,
        Boolean             indeterminate);
```

This message sets the “original” value of the object—it also sets the value stored by the superclass.

22.4.3 PrefItemGroupClass

This class acts like **GenItemGroupClass**, but has extended functionality for reading/writing strings to the .INI file. When the user triggers the dialog’s Apply trigger (when the PrefItemGroup receives a MSG_META_SAVE_OPTIONS), the PrefItemGroup will respond by writing out strings based upon which of its children items are presently selected. These strings can be taken from the items’ monikers or from strings used with **PrefStringItemClass** objects.

When loading options, if either the PIFF_USE_ITEM_MONIKERS or PIFF_USE_ITEM_STRINGS is set for the item group, then it will select its children based on the strings stored in the .INI file.Otherwise, the item group will default to the **GenItemGroupClass** functionality, which uses the identifier of the items as a basis for selection.

Code Display 22-6 PrefItemGroupClass Instance Data

```
@instance PrefInitFileFlags PIGI_initFileFlags = 0;
```

```
typedef ByteFlags PrefInitFileFlags;
#define PIFF_USE_ITEM_STRINGS 0x80
/*
 * If set, then the item group’s children must be of class PrefStringItemClass,
 * and their strings will be used to interact with the .INI file. */

#define PIFF_USE_ITEM_MONIKERS 0x40
/* If set, the monikers of the items are used to interact with the .INI file. */

#define PIFF_APPEND_TO_KEY 0x20
/* If set, the strings in this list will be ADDED
 * to strings that may already exist for this key */

#define PIFF_ABSENT_KEY_OVERRIDES_DEFAULTS 0x10
/* If set, an absent key for the item group will cause it to
 * behave as if an empty key were in the .INI file, effectively
 * overriding any default values stored with the group when it was
 * compiled. Used primarily by those groups where one subclasses
 * MSG_PREF_STRING_ITEM_CHECK_IF_IN_INIT_FILE_KEY to determine the
 * initial setting, rather than looking in the .INI file. */

@instance word PIGI_originalSelection = 0;

@instance word PIGI_originalNumSelections = 0;

@instance byte PIGI_suspendCount = 0;
/* When suspend count is nonzero, the item group won’t update text objects, nor
 * will it enable/disable objects when the selection changes. */

@vardata PrefEnableData ATTR_PREF_ITEM_GROUP_ENABLE;
/* allows setting up an object to be enabled/disabled when
 * settings are made in the item group */

typedef struct {
        word            PED_item;       /* Identifier of the item that controls
                                         * enabling/disabling of object. If this
                                         * is GIGS_NONE, then the action will be
                                         * performed if no items are selected. */
        ChunkHandle     PED_lptr;       /* Object to be enabled/disabled */
        PrefEnableFlags PED_flags;
} PrefEnableData;

typedef ByteFlags PrefEnableFlags;

#define PEF_DISABLE_IF_SELECTED 0x80
/* Disable the object if the associated item is selected,
 * otherwise do the opposite. */
```

22.4

Objects

```
#define PEF_DISABLE_IF_NONE 0x40
/* If this flag is set, then the PED_item field is ignored.
 * Instead, the item group will disable the specified object if
 * no items are selected -- or if there are no items in the list. */

@vardata word ATTR_PREF_ITEM_GROUP_OVERRIDE;
/* This attribute is used to specify an item which will act
 * as an OVERRIDE for all other items (in a non-exclusive
 * item group).
 *
 * When the specified item is SELECTED, then all other items are DE-SELECTED.
 * When any other item is SELECTED, the override item is DE-SELECTED.
 * If all items are DE-SELECTED, the override item becomes SELECTED */

@vardata ChunkHandle ATTR_PREF_ITEM_GROUP_TEXT_DISPLAY;
/* chunk handle of a GenText object that will be updated
 * with the moniker of the currently selected item on APPLY/RESET. */

@vardata ChunkHandle ATTR_PREF_ITEM_GROUP_STATUS_TEXT;
/* chunk handle of a GenText object that will be updated every
 * time this object receives a MSG_GEN_ITEM_GROUP_SEND_STATUS_MSG. */

@vardata char ATTR_PREF_ITEM_GROUP_EXTRA_STRING_SECTION[];
/* If a string is given -- the string will always be written
 * out as the FIRST string section for the .INI file key. For
 * example, the “Memory” item group uses this to always insert
 * “disk.geo” whenever writing out the memory types. Note:
 * this item is only written if the item group has either the
 * PIFF_USE_ITEM_MONIKERS or PIFF_USE_ITEM_STRINGS set. */
```

■ MSG_PREF_ITEM_GROUP_SET_ORIGINAL_SELECTION

```
void     MSG_PREF_ITEM_GROUP_SET_ORIGINAL_SELECTION(
                word selection);
```

Set the “originalSelection” field of the item group. Also, sends MSG_GEN_ITEM_GROUP_SET_SINGLE_SELECTION to superclass. This message can only be used with exclusive item groups—for other types, use the GenItemGroup messages, and then send the object MSG_PREF_SET_ORIGINAL_STATE.

Source: Anywhere.

Destination: An exclusive PrefItemGroup.

Parameters: *selection* Which item to select.

Return: Nothing.

Interception: Unlikely.

■ MSG_PREF_ITEM_GROUP_GET_SELECTED_ITEM_TEXT

```
word      MSG_PREF_ITEM_GROUP_GET_SELECTED_ITEM_TEXT(
          char   *buffer,
          word   bufSize);
```

This message returns the text in the moniker of the selected item. The value returned is the number of characters in the string. If nothing is selected, the value returned is zero.

■ MSG_PREF_ITEM_GROUP_GET_ITEM_MONIKER

```
word MSG_PREF_ITEM_GROUP_GET_ITEM_MONIKER(@stack
          char   *buffer,
          word   bufSize,
          word   identifier);
```

This message retrieves the text in the moniker of one of the items of a PrefItemGroup. This works for dynamic list subclasses as well. The message returns the number of characters in the returned text.

Parameters:

buffer	Buffer in which to return moniker text.
bufSize	Size of buffer.
identifier	Identifier of item whose moniker should be returned.

Return: Size of item's moniker.

■ MSG_PREF_ITEM_GROUP_UPDATE_TEXT

```
void MSG_PREF_ITEM_GROUP_UPDATE_TEXT(
          ChunkHandle        textObject);
```

This message updates the specified text object with text of the currently selected item. The PrefItemGroup sends this to itself in conjunction with either ATTR_PREF_ITEM_GROUP_TEXT_DISPLAY or ATTR_PREF_ITEM_GROUP_STATUS_TEXT. By default, this message will also cause the text object to send out its status message.

Source: Self.

Destination: Self.

Interception: Default behavior is to fetch the moniker of the current selection, and sent that to the text. The subclasser can use different text, if desired.

22.4.4 PrefStringItemClass

Objects of this class, used together with a PrefItemGroup with its PIFF_USE_ITEM_STRINGS flag set, allow for objects which act like GenItems but can write an arbitrary string out to the .INI file if the user selects them.

Code Display 22-7 PrefStringItemClass Instance Data

```
@class PrefStringItemClass, GenItemClass;

@instance ChunkHandle PSII_initFileString = 0;
```

■ MSG_PREF_STRING_ITEM_SET_INIT_FILE_STRING

```
void MSG_PREF_STRING_ITEM_SET_INIT_FILE_STRING(
         const char          *str);
```

This message sets the string that is read and written in the .INI file.

Interception: Unlikely but possible. Default behavior sets the string and then checks again with the .INI file to make sure item should be selected.

■ MSG_PREF_STRING_ITEM_CHECK_IF_IN_INIT_FILE_KEY

```
Boolean MSG_PREF_STRING_ITEM_CHECK_IF_IN_INIT_FILE_KEY(
         PrefItemGroupStringVars    *vars);
```

This message checks the passed string to see if the string bound to this item is in one of its pieces. A PrefItemGroup will send this object when loading options if it has the PIFF_USE_ITEM_STRINGS flag set.

Source: PrefItemGroup object.

Destination: PrefStringItem object.

Interception: May be intercepted if the subclass needs to check other things to decide whether it should be selected. If you also wish to have the default behavior, you may call the superclass either before or after you've made your own decision, as appropriate.

Parameters: *vars* Information about the string.

Return: Returns *true* (i.e. non-zero) if bound string is within the string stored in the local variable.

22.4.5 PrefBooleanGroupClass

This class behaves as GenBooleanGroup, except that it saves its original state and handles MSG_GEN_RESET appropriately. It can also use item strings or monikers when loading and saving options.

Code Display 22-8 PrefBooleanGroupClass Instance Data

```
@class PrefBooleanGroupClass, PrefClass;
@instance PrefInitFileFlags      PBGI_initFileFlags = 0;
@instance word                   PBGI_originalState = 0;
```

■ MSG_PREF_BOOLEAN_GROUP_SET_ORIGINAL_STATE

```
void MSG_PREF_BOOLEAN_GROUP_SET_ORIGINAL_STATE(
        word state);
```

This message sets the “original” state of the Boolean group.

22.4.6 PrefDynamicListClass

Objects of **PrefDynamicListClass** act like GenDynamicList objects, but with the intelligence to write their data to the .INI file. Developers will generally not use this class directory, but instead will subclass, intercepting the following messages:

MSG_PREF_DYNAMIC_LIST_BUILD_ARRAY
To build the data structures which will be used when later supplying monikers

MSG_PREF_ITEM_GROUP_GET_ITEM_MONIKER
To supply the moniker for a requested item on demand

MSG_PREF_DYNAMIC_LIST_FIND_ITEM
To find a given item when passed its moniker.

By handling these three messages, the developer will have a dynamic list that can properly load and save its selection and will scroll to the correct item in response to keyboard input.

The load and save mechanism is currently only configured for exclusive dynamic lists.

■ MSG_PREF_DYNAMIC_LIST_BUILD_ARRAY

```
void MSG_PREF_DYNAMIC_LIST_BUILD_ARRAY();
```

The PrefDynamicList sends itself this message in response to MSG_PREF_INIT.

Source: Self.

Destination: Self.

Interception: The subclasser should create an array of item strings, ideally sorted.

■ MSG_PREF_DYNAMIC_LIST_FIND_ITEM

```
 Boolean MSG_PREF_DYNAMIC_LIST_FIND_ITEM(
        word                *itemPtr,
        const char          *str,
        Boolean             ignoreCase);
```

This message finds an item given a moniker. It is normally used in the context of loading options.

Source: Self.

Destination: Self.

Parameters: *itemPtr* Pointer to space in which to return a word.

str String to search for.

ignoreCase Flag requesting a case-insensitive search.

Return: Returns *true* (i.e. non-zero) if an item was found with a moniker matching the passed string; *itmPtr* will have been filled with the number of the matching item. If no such item was found, the message will return *false* (i.e. zero) and *itmPtr* will point to the first item after the requested item.

22.4.7 TitledGlyphClass

Each TitledGlyph displays two monikers, one icon and one text. **TitledGlyphClass** has no unique instance data or messages. You are expected to provide a list of monikers in the *GI_visMoniker* field, and this list should include several icons and a text moniker. Normally the same icon is used as by the routine providing Preferences module information.

22.4

22.4.8 PrefInteractionClass

If you have any Preferences objects which write out their values to the .INI file, and if you would normally have those objects appear under a GenInteraction, then you should have them appear under a PrefInteraction instead.

Objects of the class will forward the following messages on to their children:

MSG_PREF_INIT, MSG_PREF_SET_ORIGINAL_STATE
Object will send these messages to every child that is a subclass of **PrefClass**.

MSG_META_LOAD_OPTIONS
Object will send this to each child that satisfies the appropriate criteria (based on the child's PA_LOAD_IF_USABLE flag).

MSG_META_SAVE_OPTIONS
Object will send this to each child that satisfies the appropriate criteria (based on the child's PA_SAVE_IF_CHANGED, PA_SAVE_IF_ENABLED, and PA_SAVE_IF_USABLE flags).

MSG_PREF_HAS_STATE_CHANGED, MSG_PREF_GET_REBOOT_INFO
Propagates these messages to children, returning the value from the first child that returns a non-null response.

Code Display 22-9 PrefInteractionClass Instance Data

```
@class PrefInteractionClass, PrefClass;

@instance PrefInteractionAttrs PII_attrs = 0;

typedef ByteFlags PrefInteractionAttrs;
```

Objects

```
#define PIA_LOAD_OPTIONS_ON_INITIATE 0x80
/* If set, then the dialog will send MSG_PREF_INIT,
 * followed by MSG_META_LOAD_OPTIONS to itself when it
 * receives a MSG_GEN_INTERACTION_INITIATE */

#define PIA_SAVE_OPTIONS_ON_APPLY 0x40
/* This flag is normally OFF to allow non-dialog prefInteractions to reside inside
 * other interactions without duplicate SAVE_OPTIONS messages being sent. */
```

22.4.9 PrefDialogClass

The top level object of any Preferences module UI tree should probably be a PrefDialog object. If the user's changes will necessitate a system reset, then this object will be responsible for rebooting the system.

On receiving MSG_GEN_APPLY, the dialog sends MSG_PREF_GET_REBOOT_INFO to itself (which is propagated to all children via **PrefInteractionClass**). If any children require reboot, then the dialog sends MSG_PREF_DIALOG_CONFIRM_REBOOT to itself, which normally puts up a confirmation dialog box (this can be subclassed to provide other behavior). If user confirms reboot, then the dialog will send MSG_PREF_DIALOG_REBOOT to itself, and MSG_GEN_APPLY to its superclass to save the options (it being assumed that the PIA_SAVE_OPTIONS_ON_APPLY flag is set).

Code Display 22-10 PrefDialogClass Instance Data

```
@class PrefDialogClass, PrefInteractionClass;

@default GII_visibility = GIV_DIALOG;

@default GII_type = GIT_PROPERTIES;

@default GII_attrs = @default | GIA_NOT_USER_INITIATABLE | GIA_MODAL;

@default PII_attrs = @default | PIA_SAVE_OPTIONS_ON_APPLY;
```

■ MSG_PREF_DIALOG_CONFIRM_REBOOT

```
@message Boolean MSG_PREF_DIALOG_CONFIRM_REBOOT(
         optr               string);
```

By default, the PrefDialog will respond to this message by putting up a dialog box asking the user if he wants to restart the system.

Parameters:

string Object pointer of string which will be inserted in a sentence of the form “The system software will shut down and restart to change the *string*.”

Return: Returns *true* (i.e. non-zero) to confirm the reboot; returns *false* (i.e. zero to deny the reboot.

■ MSG_PREF_DIALOG_REBOOT

```
void MSG_PREF_DIALOG_REBOOT();
```

This message causes GEOS to restart. This message does not return.

22.4.10 PrefTextClass

Objects of this class act like GenText objects, but have mechanisms to write their text string to the .INI file.

Code Display 22-11 PrefTextClass Instance Data

```
@class PrefTextClass, PrefClass;

@instance ChunkHandle PTI_originalText = 0;

@vardata ChunkHandle ATTR_PREF_TEXT_INIT_FILE_CATEGORY_TARGET;
/* PrefTextClass has the wonderful ability that it can set another
 * object’s .INI. file category from its text. Category gets updated
 * whenever the text object receives MSG_GEN_TEXT_SEND_STATUS_MSG.
 * To make this happen, put the ChunkHandle of the object in this field. */
```

22.4.11 PrefControlClass

This subclass of **PrefInteractionClass** acts like GenControl, but bypasses some GenControl functionality. This class is assumed not to be on notification lists. Objects of this class will pass MSG_GEN_RESET to their children as a normal GenInteraction would. This class has no special instance data or messages.

22.4

22.4.12 PrefTimeDateControlClass

This controller allows the user to set the system time. This class' messages and instance data fields are internal. To get and set the date and time directly, use **TimerGetDateAndTime()** and **TimerSetDateAndTime()**.

22.4.13 PrefTriggerClass

PrefTriggerClass is not a subclass of **PrefClass**; it is a subclass of **GenTriggerClass**. It acts like a normal trigger except that when activated it will send a second message if it has been set up with a ATTR_PREF_TRIGGER_ACTION field.

Code Display 22-12 PrefTriggerClass Instance Data

```
@class PrefTriggerClass, GenTriggerClass;

@vardata PrefTriggerAction ATTR_PREF_TRIGGER_ACTION;

typedef struct {
        word PTA_message;
        optr PTA_dest;
} PrefTriggerAction;
```

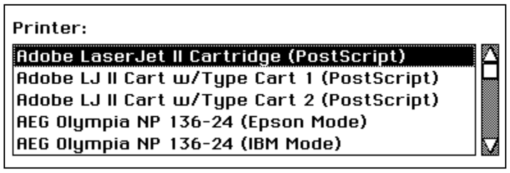
22.4.14 PrefTocListClass

Advanced Topic

22.4

PrefTocListClass allows users to choose a device driver by selecting the appropriate device name. It is assumed that each driver can control one or more types of devices. When you set up each driver, you will set up strings to describe which devices that driver can control.

This class is a fully functional subclass of **PrefDynamicListClass** (i.e. it handles those messages one must intercept to create a viable Preferences dynamic list class). It can be used to scan a list of files (not necessarily device drivers), as long as all of those files are in the same directory and have the same characters in their token.


Printer:

- Adobe LaserJet II Cartridge (PostScript)
- Adobe LJ II Cart w/Type Cart 1 (PostScript)
- Adobe LJ II Cart w/Type Cart 2 (PostScript)
- AEG Olympia NP 136-24 (Epson Mode)
- AEG Olympia NP 136-24 (IBM Mode)

Figure 22-3 *PrefTocList in Action*

Code Display 22-13 PrefTocListClass Instance Data

```
@class PrefTocListClass, PrefDynamicListClass;

@instance TocUpdateCategoryFlags PTLI_flags = 0;


typedef WordFlags TocUpdateCategoryFlags;
#define TUCF_EXTENDED_DEVICE_DRIVERS 0x8000
/* Files being enumerated are assumed to be extended device drivers. */

#define TUCF_CUSTOM_FILES 0x4000
/* The TUCP_fileArrayElementSize field will be used when creating the files array.
 * Otherwise, each element of the files array will be of size TocFileStruct.
 * NOTE: If this flag is used, the data structure used for each file
 * element MUST contain TocFileStruct as its first element. */

#define TUCF_ADD_CALLBACK 0x2000
/* TUCP_addCallback contains a fptr to a callback
 * routine that will be called when a file is added to the files array. */
```

Objects

```
#define TUCF_DIRECTORY_NOT_FOUND 0x1000
/* Don't actually scan the directory, because it doesn't exist.
 * Just create the category, and leave it empty. */

@instance TocCategoryStruct PTLI_tocInfo = {"", 0, 0};

typedef struct {
        TokenChars      TCS_tokenChars;
        DBGroupAndItem  TCS_files;      /* file name array */
        DBGroupAndItem  TCS_devices;    /* device name array--only if
                                         * TCF_EXTENDED_DEVICE_DRIVERS is set. */
} TocCategoryStruct;

@vardata PrefTocExtraEntry ATTR_PREF_TOC_LIST_EXTRA_ENTRY_1;
@vardata PrefTocExtraEntry ATTR_PREF_TOC_LIST_EXTRA_ENTRY_2;

typedef struct {
        ChunkHandle     PTEE_item;
        /* lptr of item name. For device lists, this is
         * the device. For others, this is the file name */

        ChunkHandle     PTEE_driver;
        /* Driver name (for device lists ONLY) */

        word            PTEE_info;
        /* Extra word of information */
} PrefTocExtraEntry;

@vardata char ATTR_PREF_TOC_LIST_INFO_KEY[];
/* This key is used to write the "info" word for device lists */
```

Use ATTR_GEN_PATH_DATA to tell the PrefTocList object in which directory to look for the device drivers. The PrefTocList will store the list of device drivers and device names in DB items referenced in the *PTLI_toolInfo* field's *TCS_files* and *TCS_devices* slots—it will do this automatically, so if you wish the default behavior, leave these as zero when declaring the object. If you will write your own handlers for keeping track of driver information, you may wish to work with these fields as there are some routines which have been set up to work with them.

Code Display 22-14 PrefTocList Framework

```
@object PrefTocListClass MyPrefToc = {
        PTLI_flags      = @default | TUCF_EXTENDED_DEVICE_DRIVERS;
        PTLI_tocInfo    = {"TOKN", 0, 0};
        HINT_ITEM_GROUP_SCROLLABLE;
        ATTR_GEN_INIT_FILE_KEY = "keyName";
        ATTR_GEN_PATH_DATA = { file path containing drivers };
}
```

The following routines are rather esoteric. You should only use them if you wish to subclass **PrefTocListClass**' behavior and wish to continue using its existing data structures.

TocSortedNameArrayAdd()
Use this routine to add a new name to a sorted array. The new name will be inserted in its correct place in the alphabetically sorted array.

TocSortedNameArrayFind()
This routine determines whether a given name is in the TOC List's sorted name array, and will return the position of the name's element if found.

TocFindCategory()
Use this routine to find a category in the Toc file. The *TCS_tokenChars* field of the passed **TocCategoryStruct** should already be filled in; this routine will fill in the other fields.

TocNameArrayFind()
Use this routine to get back the token associated with a name in the name array. If the name is not found, the routine will return CA_NULL_ELEMENT.

TocNameArrayGetElement()
This routine takes a name array and a token and returns information about the name array element with the given token.

TocUpdateCategory()
This complicated routine will update a category using a

callback routine which you must write. It updates the file lists by scanning the current directory.

TocNameArrayAdd()
Use this routine to add an element to a Toc name array.

TocDBLock()
Use this routine to lock one of the name arrays.

TocDBLockGetRef()
This C-only routine does the same thing as **TocDBLock()** and also returns the DB item's pointer and optr.

TocGetFileHandle()
Use this routine to get the handle of the file in which all Toc device driver information is stored.

■ MSG_PREF_TOC_LIST_GET_SELECTED_ITEM_INFO

```
Boolean MSG_PREF_TOC_LIST_GET_SELECTED_ITEM_INFO(
        word              *infoPtr);
```

This message returns the info word about the currently selected item. This message can only be used with PrefTocList objects that have the TCF_EXTENDED_DEVICE_DRIVERS flag set.

Parameters: infoPtr — Pointer to a word in which to return the info word.

Return: Returns *false* (i.e. zero) if item was found, returns *true* (i.e. non-zero) if item was not found.

■ MSG_PREF_TOC_LIST_GET_SELECTED_DRIVER_NAME

```
word MSG_PREF_TOC_LIST_GET_SELECTED_DRIVER_NAME(
        char              *buf,
        word              bufSize);
```

This message returns the name of the driver for the selected device. This message may only be used if TCF_EXTENDED_DEVICE_DRIVERS is set.

Parameters: *buf* — Pointer to buffer to fill with the driver name.

bufSize — The size of *buf*.

Return: If the device name fits in *buf*, then *buf* is filled and the message returns the size of the device name. If the device name doesn't fit, then the message will return zero.

■ MSG_PREF_TOC_LIST_GET_SELECTED_ITEM_PATH

```
void MSG_PREF_TOC_LIST_GET_SELECTED_ITEM_PATH(
         TocItemPath        *data);
```

This message returns the absolute path of the selected driver or file.

Structures:

```
typedef struct {
  word            TIP_disk; /* disk handle */
  MemHandle       TIP_pathBlock;
      /* Handle of block holding path (locked) */
  char            *TIP_path;
      /* Locked null-terminated absolute path */
} TocItemPath;
```

■ MSG_PREF_TOC_LIST_CHECK_DEVICE_AVAILABLE

```
Boolean MSG_PREF_TOC_LIST_CHECK_DEVICE_AVAILABLE(
         word              *retvalPtr);
```

This message determines if the device selected by the list is available in the machine. This message may only be called for PrefTocList objects that have the TCF_EXTENDED_DEVICE_DRIVERS flag set.

Parameters: *retValPtr* Pointer to a word which will be filled with the return value.

Return: Returns *true* (i.e. non-zero if the device is available; if the driver is a video driver, **retValPtr* is set to the device's **DisplayType**. If the device is not available, returns *false* (i.e. zero); *retValPtr* is zero of the device doesn't exist; otherwise it will be the value of the appropriate **GeodeLoadError** plus one.

VisClass

23

The visible classes—**VisClass**, **VisCompClass**, **VisContentClass**, and **VisTextClass**—can be extremely useful to application developers. They provide a flexibility of display and use that is unavailable with the generic UI objects. Among other features, visible objects provide the hooks for and basic functionality of custom drawing, custom input handling, and custom geometry management.

VisClass is the parent class of all the visible classes and all visible UI objects. **VisClass** objects, by themselves, do nothing; **VisClass** must be subclassed and the objects must be placed in a visible object tree as children of a **VisCompClass** object. Together, **VisClass** and **VisCompClass** provide a large amount of the functionality required for creating a hierarchy of non-overlapping objects.

Many applications will use both **VisClass** and **VisCompClass** as well as **VisContentClass**. Before reading this chapter, however, you should be familiar with the concepts of the generic UI and with basic GEOS applications. You should have created a simple application to get familiar with the GEOS system and Goc, and you should be relatively familiar with the GenView.

You should also become familiar with both **VisContentClass** and **VisCompClass**; both of these classes and their relationship to **VisClass** are described in the following sections. This chapter provides a starting point and most of the specifics for someone learning about the visible world. You would also do well to read “The GEOS User Interface,” Chapter 10 of the Concepts Book.

23.1 Introduction to VisClass

VisClass is one of the three visible classes you’ll have to have a working knowledge of in order to use visible objects. Both of the others, **VisCompClass** and **VisContentClass**, are its subclasses, and both are important in visible object trees. However, **VisClass** provides the heart of the visible world.

VisClass objects do nothing useful by themselves. When you subclass **VisClass** and add your own methods, however, the real power of the visible classes can be used without the programmer having to worry about the nuts and bolts of most UI issues. These classes inherently provide the following features:

Knowledge of Visible Bounds
Every visible object has an instance data field that defines the object's bounds in the GEOS graphic coordinate space. These bounds are used for many purposes, including drawing and input handling. See "Positioning Visible Objects" on page 1338.

Comprehensive Visual Updating
When a visible object is included in an object tree displayed by a GenView object, the system automatically tells the object when it must redraw itself. For example, when the view scrolls or when part of the window becomes exposed, the system will send a drawing message through the visible tree; the objects then draw themselves at the proper location in the document space, and the view window is updated. See "Drawing to the Screen" on page 1315.

Geometry Management
When visible objects are arranged as children of a VisComp or a VisContent object, they can be set up to update their size and position appropriate to the current window and view context. The VisComp and VisContent classes provide extensive and sophisticated child management functions, though custom management is also allowed and is often used. See "Positioning Visible Objects" on page 1338, especially the messages MSG_VIS_RECALC_SIZE and MSG_VIS_POSITION_BRANCH in section 23.4.3.3 on page 1343.

Object Tree Manipulation
VisClass objects can not have children, but they can be children of **VisCompClass** and **VisContentClass** objects. All of these classes support extensive tree operations including passing messages up the tree, passing messages down the tree, adding branches, removing branches, and moving branches. See "Working with Visible Object Trees" on page 1373.

Mouse Event Handling
When an input event occurs over the GenView window in which

the visible tree is displayed, that event is immediately passed on to the VisContent object. The event then travels down the visible object tree (automatically) until it is handled by some visible object. Most often, the event is handled by the leaf object with its visible bounds directly under the mouse pointer. Individual Vis objects can also grab the mouse so that they receive input events that occur even outside their bounds. See both "Handling Input" on page 1351 and "Input," Chapter 11 of the Concepts Book.

Keyboard Event Handling
A visible object can grab the keyboard input stream and receive keyboard events as they occur. Typically, certain keystrokes will be intercepted by the specific UI even if a visible object has the keyboard grab; this ensures the object won't usurp functions expected by the user from his specific UI. (The object can, however, override keystroke interception, though this is strongly discouraged.) See "Input," Chapter 11 of the Concepts Book.

Interaction with Input Hierarchies
VisClass objects have an inherent knowledge of the input hierarchies. Many visible objects may need to interact with the Focus and Target input hierarchies especially, and they can also interact with the Model hierarchy as well. See "Input," Chapter 11 of the Concepts Book.

Use of VisMonikers
Typically, visible objects will draw themselves in their entirety and will not need visual monikers. Monikers are typically labels attached to objects that get displayed either on or near the object. Visual monikers are most frequently used with generic UI objects, but they can be set up and used with visible objects as well. See "Using Visual Monikers" on page 1328.

Standard MetaClass Functions
Because **VisClass** is a subclass of **MetaClass**, it inherits all the basic, standard object functions of that class (e.g., state saving, instantiation and initialization, detaching and destruction, etc.). See "MetaClass" on page 41 of "System Classes," Chapter 1.

Many programmers new to object-oriented programming may think that visible objects are used only in certain circumstances. On the contrary, visible objects may be used for nearly any purpose and to provide nearly any application-specific graphical or user interface functions. When used properly, visible objects can relieve you of hours of writing specialized code.

Programmers new to OOP may need to practice with several small applications in order to gain experience using visible objects. You can also study the sample applications, many of which use visible objects extensively.

Visible objects are most useful when you have well-defined items that must appear on the screen or interact with the user. Pieces in a board game provide a good example: Each piece has its own visual representation, and each piece has a special set of behaviors that depend on user actions and game context. Therefore, it makes sense to have each piece be an object of **VisClass**.

Another, less immediately obvious, example might be a program that represents maps on the screen. The map itself could be stored as a visible composite object (**VisCompClass**), and each of the major subdivisions on the map could be a visible object. For example, in a map of the United States, the country would be a VisComp object, and each state could be a Vis object. Taken one step further, each state could be a composite object, and each county in the state a visible child of the state. Another step turns the counties into composite objects and each town a visible object. This arrangement lends itself to a tree structure, which is how visible objects are stored.

Another example is a spreadsheet application. The spreadsheet itself could be one composite object containing a number of row objects. Each row object is a composite that holds a number of cell objects. Each cell is a visible object with the functions of the cell built into it. The row and spreadsheet composites can use the geometry manager to arrange the cells properly, thereby making the programmer free of ever having to worry where the cells are in the document. The visible object tree will automatically pass input events to the cell under the mouse pointer. Entire rows can be created, destroyed, drawn, erased, or moved with a few simple commands by the programmer.

Objects

23.2 The Visible Class Tree

See Also

"The GEOS User Interface," Chapter 10 of the Concepts Book, also describes visible classes and visible trees.

As stated earlier, the visible class tree consists of four classes. The relationship between these classes is shown in Figure 23-1.

VisClass This is the parent class of the visible tree. It contains the basic functionality common to every visible object and is the subject of this chapter.

VisCompClass
A subclass of **VisClass**, this class has the added ability to have and arrange children in the visible tree. It has sophisticated geometry management functions built in and can be used as an organizational object.

VisContentClass
A subclass of **VisCompClass**, this class is used only as the content of a GenView. The VisContent has default handlers for many messages sent out by the view, and it interacts with the view window to display the visible object tree. Every visible object tree, to be displayed in a GenView, must have a VisContent as its root object.

VisTextClass
This class, not discussed in this chapter, is a visible version of the **GenTextClass**. It provides a nearly full-featured word processing object that can be put in the visible tree.

The first three of these classes are used extensively by many applications.

23.3 VisClass Instance Data

VisClass has seven different instance data fields as shown in Code Display 23-1. Each of these fields is discussed in detail in the following sections, and each may be set or reset during execution by sending the visible object certain messages.

Several of the instance fields will never be used by the application programmer. Most likely, you will work directly with the *VI_bounds* field and

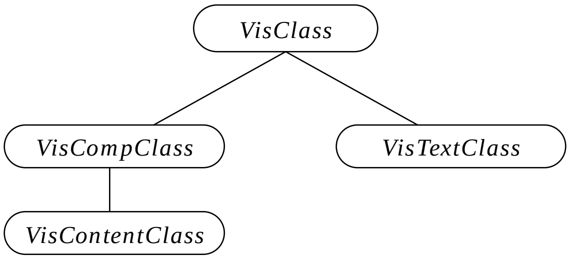


Figure 23-1 *Visible Class Tree*

All the classes shown have the properties of VisClass and can be used in an application's visible object tree. Visible objects are more flexible but require more programming than do generic objects.

perhaps the *VI_attrs* or *VI_geoAttrs* fields. It's unlikely you will need to use the others unless you're doing something quite unusual.

Code Display 23-1 VisClass Instance Data

```
/* This is the definition of the VisClass instance data fields with some comments
 * as to their use. */

        /* VI_bounds
         * VI_bounds is a Rectangle structure defining the outermost boundaries of
         * the visible object. */

    @instance Rectangle         VI_bounds = {0, 0, 0, 0};

        /* VI_typeFlags
         * VI_typeFlags is a record containing eight flags that determine the
         * type of the visible object. None are set by default. */

    @instance VisTypeFlags      VI_typeFlags = 0;
        /* Possible flags:
         * VTF_IS_COMPOSITE             VTF_IS_WINDOW
         * VTF_IS_PORTAL                VTF_IS_WIN_GROUP
         * VTF_IS_CONTENT               VTF_IS_INPUT_NODE
         * VTF_IS_GEN                   VTF_CHILDREN_OUTSIDE_PORTAL_WIN */

        /* VI_attrs
         * VI_attrs is a record that contains eight attribute flags for the
         * object. Those set by default are shown. */
```

```
@instance VisAttrs           VI_attrs = (VA_MANAGED | VA_DRAWABLE |
                                        VA_DETECTABLE | VA_FULLY_ENABLED);
    /* Possible flags:
     * VA_VISIBLE                    VA_FULLY_ENABLED
     * VA_MANAGED                    VA_DRAWABLE
     * VA_DETECTABLE                 VA_BRANCH_NOT_MINIMIZABLE
     * VA_OLD_BOUNDS_SAVED           VA_REALIZED */

    /* VI_optFlags
     * VI_optFlags is a record containing eight window update flags.
     * The flags set by default are shown. */

@instance VisOptFlags        VI_optFlags = (VOF_GEOMETRY_INVALID |
                                 VOF_GEO_UPDATE_PATH | VOF_WINDOW_INVALID |
                                 VOF_WINDOW_UPDATE_PATH |
                                 VOF_IMAGE_INVALID |
                                 VOF_IMAGE_UPDATE_PATH);
    /* Possible flags:
     * VOF_GEOMETRY_INVALID          VOF_GEO_UPDATE_PATH
     * VOF_IMAGE_INVALID             VOF_IMAGE_UPDATE_PATH
     * VOF_WINDOW_INVALID            VOF_WINDOW_UPDATE_PATH
     * VOF_UPDATE_PENDING            VOF_EC_UPDATING */

    /* VI_geoAttrs
     * VI_geoAttrs is a record of eight geometry management flags. */

@instance VisGeoAttrs        VI_geoAttrs = 0;
    /* Possible flags:
     * VGA_GEOMETRY_CALCULATED       VGA_NO_SIZE_HINTS
     * VGA_NOTIFY_GEOMETRY_VALID     VGA_DONT_CENTER
     * VGA_USE_VIS_SET_POSITION      VGA_USE_VIS_CENTER
     * VGA_ALWAYS_RECALC_SIZE        VGA_ONLY_RECALC_SIZE_WHEN_INVALID

    /* VI_specAttrs
     * VI_specAttrs is a record containing eight flags used when the visible
     * object is part of a specific UI library. This field is rarely if ever
     * used by applications. */

@instance SpecAttrs          VI_specAttrs = 0;
    /* Possible flags:
     * SA_ATTACHED                   SA_REALIZABLE
     * SA_BRANCH_MINIMIZED           SA_USES_DUAL_BUILD
     * SA_CUSTOM_VIS_PARENT          SA_SIMPLE_GEN_OBJ
     * SA_CUSTOM_VIS_PARENT_FOR_CHILD
     * SA_TREE_BUILT_BUT_NOT_REALIZED */
```

```
        /* VI_link
         * VI_link contains an object pointer to the visible object’s next sibling
         * in the tree. Note that VisClass has no corresponding @composite field;
         * if the object is to have children, it must be of class VisCompClass.

    @instance @link             VI_link;
```

23.3

23.3.1 VI_bounds

The *VI_bounds* field is a **Rectangle** structure containing four values: The left, top, right, and bottom outer bounds of the object. The bounds of a visible object describe the object to the rest of the world: the input manager and UI use them to determine when the mouse pointer is over the object, and the object’s visible parent (if any) uses them to figure out the total geometry of all its children.

The *VI_bounds* field is very important to nearly all visible objects. In fact, section 23.4.3 on page 1338 is devoted to this field and how it is used. For full information on visible bounds, see that section.

23.3.2 VI_typeFlags

VI_typeFlags, MSG_VIS_SET_TYPE_FLAGS,
MSG_VIS_GET_TYPE_FLAGS

The *VI_typeFlags* field is a bitfield record containing eight flags. These flags determine the type of the visible object and are used primarily by the UI and the windowing system. When used at all, these flags are typically set in the object’s definition or when the object is first created, and they are rarely changed dynamically by an application. The eight flags in this record are listed below. The record is of type **VisTypeFlags**.

VTF_IS_COMPOSITE

This flag is set if the object is of **VisCompClass** rather than **VisClass**. This flag indicates the object may have children (normal **VisClass** objects can not have children). Because this is not the only difference between these classes, changing this flag can have unpredictable results.

VTF_IS_WINDOW
This flag is set if the object creates a window via the window system in order to display itself and its children. This flag implies the object must also have VTF_IS_COMPOSITE set, meaning the object must actually be of **VisCompClass** and not **VisClass**. If this flag is set, the UI will assume that the *VI_bounds* field of the object represents the current size of the window. Objects with the VTF_IS_WINDOW flag set are skipped by the normal input and output messages such as MSG_VIS_DRAW and MSG_META_START_SELECT; these messages will be passed directly to the object's appropriate child. VTF_IS_WINDOW and VTF_IS_PORTAL are mutually exclusive.

VTF_IS_PORTAL
This flag is set if the object is a portal. A portal is an object that has its own window and appears in its parent's window. A portal differs from a window (VTF_IS_WINDOW) in that the window has no part of itself drawn in its parent window. This flag implies that the VTF_IS_COMPOSITE flag must also be set indicating the object may have visible children. All visible children of a portal will appear in the portal's associated window if VTF_CHILDREN_OUTSIDE_PORTAL_WIN is set. The object itself may display portions of itself within its parent's window, though; an example of this is the UI's implementation of the GenDisplayControl object, which manages several child windows within its own window area. Its border is drawn in the parent window, then its window is defined to be one pixel inside this border. VTF_IS_PORTAL and VTF_IS_WINDOW are mutually exclusive.

VTF_IS_WIN_GROUP
This flag is set for the top visible object in a visible branch which makes that branch a realizable entity. This flag implies that both VTF_IS_COMPOSITE and VTF_IS_WINDOW must be set as well. An example would be the specific UI's implementation of a GenPrimary object—if the primary is not visible, none of its children may be visible (with some exceptions).

VTF_IS_CONTENT
This flag is set if the object is the output of a window object. Typically, this will be set only if the object is of

VisContentClass (not for **VisClass** or **VisCompClass**). Note that if this is set, both VTF_IS_WINDOW and VTF_IS_WIN_GROUP must also be set.

VTF_IS_INPUT_NODE

This flag is an optimization bit. It is set if the object controls input flow (such as a VisContent) or is a target or focus node that is not a window group (VTF_IS_WIN_GROUP). MSG_VIS_VUP_ALTER_INPUT_FLOW is sent directly to objects with this bit set unless there is a need for them to actually use the VUP mechanism, as is the case with mouse grabs in a 32-bit content. Similarly, MSG_META_MUP_ALTER_FTVMC_EXCL, MSG_VIS_FUP_QUERY_FOCUS_EXCL, and MSG_VIS_VUP_QUERY_FOCUS_EXCL are sent to the first visible object up the tree that either is a window group or has this bit set.

VTF_IS_GEN

This flag is set if the visible object is the specific UI counterpart to a generic object. This flag must be set for the object to handle messages that begin MSG_SPEC. The only objects that will ever have this flag set are those in Specific UI libraries.

VTF_CHILDREN_OUTSIDE_PORTAL_WIN

This flag is set if children of a portal object may appear in the object's parent window rather than in the portal's specially-created window. The VTF_IS_PORTAL flag must also be set for this object. An example of this is the specific UI's representation of a GenView object and its children; the view's children appear along the edges of the view window rather than inside it. The GenDisplayControl, by contrast, has all its children appearing within its window; thus, it does not have this flag set.

Although the *VI_typeFlags* record will rarely be changed at run-time, you can set new values by sending MSG_VIS_SET_TYPE_FLAGS to the visible object while it is realized. To retrieve the current type flags, send the object a MSG_VIS_GET_TYPE_FLAGS.

■ MSG_VIS_SET_TYPE_FLAGS

```
void        MSG_VIS_SET_TYPE_FLAGS(
            VisTypeFlags        flagsToSet,
            VisTypeFlags        flagsToClear);
```

This message causes the object to set its *VI_typeFlags* field to the values passed. The *flagsToSet* parameter will be OR-ed with *VI_typeFlags*, and the *flagsToClear* parameter will be logically inverted and then AND-ed with the field. The clear operation will occur after the set. Therefore a flag set in both parameters will be cleared by this message.

Source: Unrestricted.

Destination: Any visible object.

Parameters: *flagsToSet* A record of **VisTypeFlags** indicating which flags are to be set for the object.

flagsToClear A record of **VisTypeFlags** indicating which flags are to be cleared for the object (a flag set in *flagsToClear* will be cleared in *VI_typeFlags*).

Return: Nothing.

Interception: Unlikely.

■ MSG_VIS_GET_TYPE_FLAGS

```
VisTypeFlags MSG_VIS_GET_TYPE_FLAGS();
```

This message returns a visible object's current *VI_typeFlags* field.

Source: Unrestricted.

Destination: Any visible object.

Parameters: None.

Return: A byte of **VisTypeFlags** representing the object's current *VI_typeFlags* field.

Interception: Unlikely.

23.3.3 VI_attrs

`VI_attrs, MSG_VIS_GET_ATTRS, MSG_VIS_SET_ATTRS`

The *VI_attrs* field is a record of eight flags (**VisAttrs**)that determine the general visible attributes of the object. These attributes may be changed on the fly and often are. Applications that don't involve visible tree operations (adding, removing, or moving branches) probably will not change these attributes dynamically.

To set the flags after the object has been instantiated, send it a MSG_VIS_SET_ATTRS with the appropriate flags to be set. To retrieve the *VI_attrs* record, send the object a MSG_VIS_GET_ATTRS. The VA_REALIZED flag may not be changed with MSG_VIS_SET_ATTRS.

The eight flags of the **VisAttrs** record are listed below:

VA_VISIBLE
This flag is effective only for window group objects (those with VTF_IS_WIN_GROUP set in *VI_typeFlags*) and is ignored for other objects. If set, it indicates that the visible branch headed by this object can be made visible as soon as it is linked visibly to another visible branch. For example, a window group object may have this bit set and not be on the screen if its parent window group is not set VA_VISIBLE; as soon as the parent window group becomes VA_VISIBLE, though, the entire branch will become visible.

VA_FULLY_ENABLED
This flag is normally set for visible objects. If it is set, the object is fully enabled when visible on the screen. If not set, the object is not fully enabled—the object will typically be drawn in a 50% pattern (grayed out) and will not receive click events. This flag works even for normal visible objects, though this behavior is seen almost exclusively with generic UI objects.

VA_MANAGED
This flag indicates that the object's location is managed by its parent. If the flag is set, space is set aside in the parent composite by the geometry manager. When the window displaying this object changes, the geometry will have to be calculated. If this flag is not set, the object can alter its own location and bounds without affecting the parent's geometry.

VA_DRAWABLE
This flag indicates whether the object is drawn or invisible. If the flag gets set or reset during execution, the rectangle defined by its *VI_bounds* field will automatically be marked invalid.

VA_DETECTABLE
This flag, when set, indicates that the object might respond to mouse, pointer, keyboard, or other similar events. If the flag is reset (such as with display-only objects), its parent composite object will not bother passing it these events. This flag must be set if the object is a window composite. If the status of this flag is changed during execution, the object's grab state will not change. For example, if the object has the mouse grab when a MSG_VIS_SET_ATTRS resets the flag, the object will retain the mouse grab until it is released or preempted normally.

VA_BRANCH_NOT_MINIMIZABLE
This flag is used *only* by the visible counterparts of generic UI objects. If set, it indicates that this branch (e.g., a GenInteraction and its children) stays visible and usable even when the parent primary window is minimized.

VA_OLD_BOUNDS_SAVED
This flag, when set, causes the geometry manager to store the object's old bounds when its *VI_bounds* field changes. The old bounds are stored in a TEMP_VIS_OLD_BOUNDS vardata field.

VA_REALIZED
This flag indicates whether the object is visibly open on the screen—i.e., whether it is positioned on an open window. If the flag is set, the object is visible on the screen. When the object is no longer visible on the screen, this flag is reset. This flag may *not* be altered by MSG_VIS_SET_ATTRS; it may only be changed by MSG_VIS_OPEN (which sets the flag) and MSG_VIS_CLOSE (which resets it).

■ MSG_VIS_SET_ATTRS

```
void    MSG_VIS_SET_ATTRS(
        VisAttrs          attrsToSet,
        VisAttrs          attrsToClear,
        VisUpdateMode     updateMode);
```

This message sets the object's *VI_attrs* field to the values passed. The *attrsToSet* parameter will be OR-ed with *VI_attrs*, and the *attrsToClear* parameter will be logically inverted and then AND-ed with the field. The clear operation will occur after the set, and therefore a flag set in both parameters will be cleared. The *updateMode* parameter indicates when the visual update of the object should occur after the new attributes have been set.

Source: Unrestricted.

Destination: Any visible object.

Parameters: *attrsToSet* A record of **VisAttrs** indicating which flags are to be set for the object.

attrsToClear A record of **VisAttrs** indicating which flags are to be cleared for the object (a flag set in *attrsToClear* will be cleared in *VI_attrs*).

updateMode A **VisUpdateMode** indicating when the object should be visually updated on the screen.

Return: Nothing.

Interception: Unlikely.

Warnings: Not all flags in the *VI_attrs* record may be set with this message.

■ MSG_VIS_GET_ATTRS

```
VisAttrs MSG_VIS_GET_ATTRS();
```

This message returns a visible object's current *VI_attrs* field.

Source: Unrestricted.

Destination: Any visible object.

Parameters: None.

Return: A byte of type **VisAttrs** representing the current flags set in the object's *VI_attrs* field.

Interception: Unlikely.

23.3.4 VI_optFlags

```
VI_optFlags, MSG_VIS_GET_OPT_FLAGS
```

The *VI_optFlags* field is a **VisOptFlags** record containing eight flags. These flags are set and used by the geometry manager to mark which objects need to be updated and how. The flags can not be set individually by applications;

however, they can be retrieved with the message MSG_VIS_GET_OPT_FLAGS. It is unlikely that your code will ever access these flags directly. The eight flags in this field are listed below.

VOF_GEOMETRY_INVALID
This flag indicates that the general geometry of this object has become invalid (with a MSG_VIS_MARK_INVALID). This is used primarily when an object knows that its bounds must be recalculated or re-verified and that any change must be reflected in its parent window. This flag will be reset when a geometry update resolves the object's bounds. Typically, this will occur with a window update message such as MSG_VIS_UPDATE_WIN_GROUP.

VOF_GEO_UPDATE_PATH
This flag, also set by MSG_VIS_MARK_INVALID, marks a trail of objects that all must have their geometry updated.

VOF_IMAGE_INVALID
This flag indicates that the image of the object or the bounds of a windowed object have changed and therefore need to be updated. This flag is set by MSG_VIS_MARK_INVALID. The image redraw will occur the next time one of the window update messages is used on the object's visual branch. This flag will be reset when a visual update redraws the object. Typically, this will occur with MSG_VIS_UPDATE_WIN_GROUP or another window update message.

VOF_IMAGE_UPDATE_PATH
This flag, also set by MSG_VIS_MARK_INVALID, marks a trail of objects that must have their images updated.

VOF_WINDOW_INVALID
This flag indicates that the window the object resides in needs to be opened or closed, or that the object itself needs to be opened or closed via MSG_VIS_OPEN or MSG_VIS_CLOSE. It is set by MSG_VIS_MARK_INVALID. This flag will be reset when a visual update occurs on the window group. Typically, this will occur with a window update message such as MSG_VIS_VUP_UPDATE_WIN_GROUP. (This message should be called whenever an object is added to a visible tree; it ensures the object will be properly opened.)

VOF_WINDOW_UPDATE_PATH
This flag, also set by MSG_VIS_MARK_INVALID, marks a trail of window objects that all must have their windows and open/closed state updated.

VOF_UPDATE_PENDING
This flag indicates that the object has an unprocessed MSG_VIS_UPDATE_WIN_GROUP in its event queue.

VOF_EC_UPDATING
This flag is used by the **VisClass** error-checking code only. It indicates that a visible branch is currently being updated; the system checks it to make sure nested updates do not occur.

■ MSG_VIS_GET_OPT_FLAGS

```
VisOptFlags MSG_VIS_GET_OPT_FLAGS();
```

This message returns the current *VI_optFlags* field of the object.

Source: Unrestricted.

Destination: Any visible object.

Parameters: None.

Return: A byte record of **VisOptFlags** reflecting the current settings in the object's *VI_optFlags* field.

Interception: Unlikely.

See Also: MSG_VIS_MARK_INVALID.

23.3.5 VI_geoAttrs

VI_geoAttrs, MSG_VIS_SET_GEO_ATTRS, MSG_VIS_GET_GEO_ATTRS

The *VI_geoAttrs* field is a **VisGeoAttrs** record of eight flags that determine the state of the object's geometry. They can be set and reset during execution to dynamically change the geometry behavior of the object. None of the flags is set by default. To set the flags, send the object a MSG_VIS_SET_GEO_ATTRS. The flags can be retrieved with a MSG_VIS_GET_GEO_ATTRS. The eight flags in this record are listed below.

VGA_GEOMETRY_CALCULATED
This flag indicates whether the object has ever been calculated.

If the flag is clear, the object's geometry is about to be calculated. An object may use this flag if it has a particular desired size. This flag is cleared before a geometry recalculation by MSG_VIS_RESET_TO_INITIAL_SIZE.

VGA_NO_SIZE_HINTS

This flag is used by the Specific UI library to determine whether the object has any sizing hints or not. If the flag is not set, the object has one or more sizing hints (e.g., HINT_INITIAL_SIZE or HINT_FIXED_SIZE).

VGA_NOTIFY_GEOMETRY_VALID

If this flag is set, the geometry manager will notify the object when its geometry messages have all been handled and the object's geometry is set. Notification will be in the form of MSG_VIS_NOTIFY_GEOMETRY_VALID. This flag may be used by visual objects for calculations or other sizing once geometry is redone but before redrawing occurs.

VGA_DONT_CENTER

This flag, when set, allows the object to individually override its parent's centering behavior. The object will instead appear at either the left edge of a vertical composite or the top edge of a horizontal composite.

VGA_USE_VIS_SET_POSITION

This flag is set to indicate that the object does not use the default handlers for MSG_VIS_SET_POSITION and MSG_VIS_POSITION_BRANCH. This flag provides an optimization that allows static calls to the geometry manager.

VGA_USE_VIS_CENTER

If this flag is set, the geometry manager will use the standard **VisClass** or **VisCompClass** center message to calculate the object's center. Unless an object is using some unusual centering behavior, it should have this flag set. This flag allows the geometry manager to speed up calculations for normal center behavior.

VGA_ONLY_RECALC_SIZE_WHEN_INVALID

This flag indicates that the object's size should be recalculated only when its geometry is marked invalid. That is, once the object's geometry is calculated, it will always return the same size until its geometry is once again marked invalid. Because the geometry manager often queries objects for their sizes, most objects should have this flag set. It allows the geometry manager to speed up calculations.

VGA_ALWAYS_RECALC_SIZE
This flag avoids optimizations that do not recalculate the object's size when passed its current size. Though not often used, this flag may be needed in certain situations (e.g., when a composite object must expand to fit and center its children).

■ MSG_VIS_SET_GEO_ATTRS

```
void      MSG_VIS_SET_GEO_ATTRS(
          VisGeoAttrs        attrsToSet,
          VisGeoAttrs        attrsToClear,
          VisUpdateMode      updateMode);
```

This message causes the object to set its *VI_geoAttrs* field to the values passed. The *attrsToSet* parameter will be OR-ed with *VI_geoAttrs*, and the *attrsToClear* parameter will be logically inverted and then AND-ed with the field. The clear operation will occur after the set, and therefore a flag set in both parameters will end up cleared. Many of the attributes in this record are used only by Specific UI libraries and should not be altered by applications.

Source: Unrestricted.

Destination: Any visible object.

Parameters: *attrsToSet* A record of **VisGeoAttrs** indicating which flags are to be set for the object.

attrsToClear A record of **VisGeoAttrs** indicating which flags are to be cleared for the object (a flag set in *attrsToClear* will be cleared in *VI_geoAttrs*).

updateMode A **VisUpdateMode** indicating when the object should be visually updated on the screen.

Return: Nothing.

Interception: Unlikely.

■ MSG_VIS_GET_GEO_ATTRS

```
VisGeoAttrs MSG_VIS_GET_GEO_ATTRS();
```

This message returns a visible object's current *VI_geoAttrs* field.

Source: Unrestricted.

Destination: Any visible object.

Parameters: None.

Return: A record of type **VisGeoAttrs** reflecting the flags currently set in the object's *VI_geoAttrs* field.

Interception: Unlikely.

23.3.6 VI_specAttrs

```
VI_specAttrs, MSG_SPEC_SET_ATTRS, MSG_SPEC_GET_ATTRS
```

The *VI_specAttrs* field is a record of eight flags. This field is accessed only by objects that are part of a Specific UI library. Applications will almost never access this field or use its flags. Documentation on creating specific UI libraries may be published under separate cover.

■ MSG_SPEC_SET_ATTRS

```
void    MSG_SPEC_SET_ATTRS(
        SpecAttrs           attrsToSet,
        SpecAttrs           attrsToClear,
        VisUpdateMode       updateMode);
```

This message sets the current *VI_specAttrs* field to the values passed. A flag set in both attributes parameters will be cleared.

Source: Unrestricted.

Destination: Any visible object used as a specific UI object.

Parameters: *attrsToSet* A record of **SpecAttrs** indicating which flags are to be set in the object's *VI_specAttrs* field.

attrsToClear A record of **VisAttrs** indicating which flags are to be cleared for the object (a flag set in *attrsToClear* will be cleared in *VI_specAttrs*).

updateMode A **VisUpdateMode** indicating when the object should be visually updated on the screen.

Return: Nothing.

Interception: Unlikely.

■ MSG_SPEC_GET_ATTRS

```
SpecAttrs MSG_SPEC_GET_ATTRS();
```

This message returns a visible object's current *VI_specAttrs* field. This field is used only by Specific UI libraries and is meaningful only for objects in these libraries.

Source: Unrestricted.

23.4

Destination: Any visible object that is used as a specific UI object.

Parameters: None.

Return: A byte of type **SpecAttrs** indicating which flags are set in the object's *VI_specAttrs* field.

Interception: Unlikely.

23.3.7 VI_link

```
VI_link
```

The *VI_link* field holds the optr of the visible object's next sibling in the visible object tree. This field may not be accessed directly by applications but may be set or changed with the various visible tree messages. For full information on how to build and manipulate visible object trees, see section 23.5 on page 1373.

23.4 Using VisClass

As stated earlier, visible objects are nearly useless unless you subclass one of the visible classes and handle certain messages. Although there's a lot of functionality built into **VisClass** and its subclasses, they must be customized to be of any use. In addition, visible objects must be put into a visible tree and connected to a GenView or other windowed UI object to be displayed.

23.4.1 Basic VisClass Rules

To use visible objects, you must subclass the Vis classes and create a visible object tree. Otherwise, the visible objects will not be able to display themselves or accept user input, the two things that make visible objects worthwhile.

Nearly all visible objects you use will at least display themselves on the screen. Some may accept mouse or keyboard input; some may move themselves around the screen, resize themselves, or implement custom geometry management functions.

For a subclass of **VisClass** to display itself on the screen, it must handle the message MSG_VIS_DRAW. This message is sent to all visible objects in a given window when that window is subjected to an exposure event. In addition, the object must be part of a tree of visible objects, the top node of which is connected to a GenView or other windowed object. Displaying a basic visible object is discussed in “Basic VisClass Rules” on page 1313 and “Drawing to the Screen” on page 1315.

For a visible object to handle mouse input, it will need to handle a subset of the mouse event messages. Typically, a visible object will want to know when the mouse is clicked within the object’s bounds (MSG_META_START_SELECT) and when the mouse button is released (MSG_META_END_SELECT). Other messages dealing with mouse motion or other mouse buttons may also be of interest to the object. In addition, visible objects may request and handle Ink input. Handling of these messages is discussed in “Handling Input” on page 1351.

Most applications using visible object trees will add objects to and remove them from the trees. MSG_VIS_ADD_CHILD and MSG_VIS_REMOVE are the two messages used most often for tree management. When an object should be freed, it can be destroyed with MSG_VIS_DESTROY, which will remove it from the tree and then free it.

Some applications will want to use the built-in geometry management features of GEOS. The geometry manager can automatically resize and reposition an entire visible object tree properly according to pre-set constraints. (The constraints can also be changed at run-time.) How the system manages visible object geometry is discussed in “Geometry Management” on page 1343.

The geometry manager, though extremely useful for non-overlapping objects, may not be sufficient for all the needs of a complex application. For an object to determine its own size and position, it has to handle some of the messages sent out by the geometry manager. You can also manually set the size and position of each visible object. This is discussed in "Positioning Visible Objects" on page 1338.

Often applications may need to change something about a visible object or the tree it's in. When this happens, the application must force a visual update by invalidating either the geometry or the image (or both) of the object and then calling MGS_VIS_VUP_UPDATE_WIN_GROUP. To mark any object invalid, the application must call MSG_VIS_MARK_INVALID.

Many visible objects will have specific functions they perform. For example, if the user presses on a menu item labeled "New Game" or something similar, the visible object may return itself to its original location (as in the TicTac sample application). To get this type of functionality, you must define new messages for your subclass of **VisClass** and have the object handle them. In the case of the TicTac game pieces, each piece handles the MSG_PIECE_NEW_GAME message by resetting the object's position to its original position.

Visible objects are maintained and managed in a tree structure. The tree has three basic elements: The root (topmost) node must be an object of **VisContentClass**. Any nodes in the middle of the tree, nodes that are allowed to have children, must be of **VisCompClass**. Any leaf nodes (guaranteed not to have children) may be of either **VisClass** or **VisCompClass**. (Subclasses of the above classes may also be used.)

The visible object tree is displayed normally through a GenView object. The output of the GenView must be tied directly to the top node of the visible object tree; this is why the visible tree must be headed by a VisContent—only the VisContent has the methods to handle GenView messages.

The VisContent object may or may not do things other than manage the visible tree. For example, the content may draw some background graphics to the window before the other visible objects draw themselves, or it may implement some special geometry behavior to arrange its children.

The VisContent can have children. These children can be either standard visible objects or *composite* visible objects. Composites, like the content, are allowed to have children. For efficiency, standard visible objects can not have children. Therefore, only leaf objects in the tree may be of **VisClass**.

The visible object tree may have any number of layers of composites. You should pay attention to the functions required in your application and fit the structure of the tree to the application's needs.

These restrictions, combined with the link to the generic object tree through GenView, result in an object tree structure somewhat like that shown in Figure 23-2. Of course, the actual generic and visible tree structure will vary from application to application. Also, note that the generic and visible sections are actually two different object trees; they are not connected through a parent–child relationship anywhere.

23.4.2 Drawing to the Screen

One of the main features of visible objects is their ability to draw themselves on the screen. The visible tree does not have to detect when drawing or

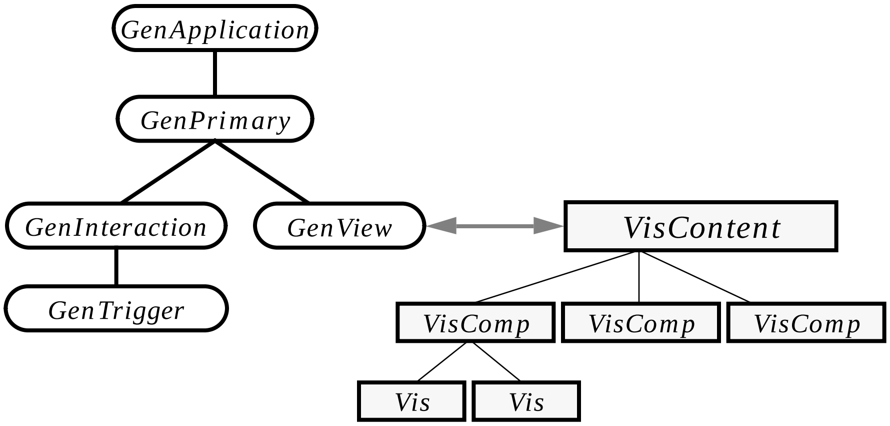


Figure 23-2 *Visible and Generic Object Trees*
The visible object tree is linked to the generic object tree through the GenView–VisContent link. VisComp objects may have children; Vis objects may not. (Each object shown simply has the suffix Class removed from its class type.)

redrawing should be done; the UI will do this automatically through the window system and the GenView object.

23.4.2.1 Visual Updates

MSG_VIS_OPEN, MSG_VIS_CLOSE

The visible tree will have to draw or redraw itself when a portion of its GenView window becomes newly exposed. The GenView keeps track of its window and will notify its content when any portion of the window becomes exposed. Exposure can occur from several events: The view window could be newly created; another window could have been moved, exposing the view's window; or the view window could have been scrolled.

Each of these exposure events looks exactly the same to the visible tree. When one of them occurs, the GenView will send a MSG_META_EXPOSED to its content object. The content (a VisContent or GenDocument if you're using a visible tree) will create a GState for the window and translate the MSG_META_EXPOSED into a MSG_VIS_DRAW. It will then send this MSG_VIS_DRAW to itself.

The default handler for MSG_VIS_DRAW in a composite is to simply pass the message on to all of the composite's children. There is no default behavior for MSG_VIS_DRAW in **VisClass**.

Any object (content, composite, or leaf) that wants to represent itself on the screen must handle MSG_VIS_DRAW. Any composite that subclasses this message must be sure to pass it on to all of its children with the following line:

```
    @send @visChildren::MSG_VIS_DRAW(drawFlags, gstate);
```

Redrawing is not initiated only by the view window becoming exposed. It can also be initiated by some portion of the visible tree changing so that it needs to be redrawn (e.g., a child is moved from one composite to another, which may have an effect on the visual representation of the composite). Typically, any change that necessitates a redrawing will automatically generate a MSG_VIS_DRAW to the appropriate objects. These operations, however, will require a **VisUpdateMode** to be passed along with the other parameters.

All of the above assumes that the visible object is "open" or "realized." If an object is not open, it will not appear on the screen or as part of a composite's geometry calculations. When a window is first created, the window's content is automatically opened with MSG_VIS_OPEN. In the case of a visible tree attached to a GenView, this means that the VisContent object will be opened along with the window. All of the VisContent's children will also be opened automatically (assuming their VOF_WINDOW_INVALID bits are set, which is true the first time a visible object comes up).

If a visible object is added to an already-opened composite, the child will automatically be opened and marked invalid. This will cause the composite's geometry and image to be updated during the next visual update. If a child is added to a composite that is not currently open, the child will not be opened.

Although it is rarely done directly, you can manually open a visible object with the message MSG_VIS_OPEN, and you can "close" an object (remove it from the screen and from the tree's geometry calculations) with MSG_VIS_CLOSE. To open or close an entire branch of the visible tree, you can send the appropriate message to the top composite object in the branch. The message will propagate down through all the visible children.

■ MSG_VIS_OPEN

```
void        MSG_VIS_OPEN(
            WindowHandle window);
```

This message is part of the visual update mechanism and will be sent by the system when an object must be visually updated to appear on the screen. It will be sent to any VOF_WINDOW_INVALID objects in a branch whose VTF_IS_WIN_GROUP object has become VA_VISIBLE. Any object added into a visible tree having VOF_WINDOW_INVALID set will receive a MSG_VIS_OPEN at the next visual update. This message propagates down the visible tree and does *not* cross over boundaries of window groups. This message is often subclassed by objects that want to initialize some information before being drawn on the screen. If an object subclasses this message, it should be sure to call its superclass somewhere in the handler.

Source: Visual update mechanism.

Destination: Any visible object.

Parameters: *window* Handle of window in which the object is to appear, or zero if it should appear in the top window object.

Return: Nothing.

Interception: Will be intercepted by objects that want to initialize certain data or take certain actions before being drawn on the screen. Any intercepting objects must call the superclass somewhere in their handler.

■ MSG_VIS_CLOSE

```
void     MSG_VIS_CLOSE();
```

This message is part of the visual update mechanism and is sent to objects that are being taken off the screen. The system will send this message to any objects in a visual branch whose VTF_IS_WIN_GROUP object is set not visible (~VA_VISIBLE). This message closes appropriate windows and propagates to all children to the bottom leaf objects of the visible branch. This message will *not* cross window group boundaries; if a lower VTF_IS_WIN_GROUP object is encountered during the message propagation, an error will likely occur. Therefore, lower window groups must be closed before higher window groups may be closed.

Source: Visual update mechanism.

Destination: Any visible object.

Interception: Will be intercepted by objects that want to do something as it is being taken off the screen. Any objects that intercept the message must call their superclasses in the handlers.

23.4.2.2 VisUpdateMode

VisUpdateMode is an enumerated type that determines when and how the visible tree will be visually updated when its image or geometry becomes invalid. This type has four enumerations, described below:

VUM_MANUAL
This mode indicates that the UI should not update at all after this change. Instead, the visual update will be initiated manually later on.

VUM_NOW
This mode indicates that the visual update should occur immediately after the change is effected.

VUM_DELAYED_VIA_UI_QUEUE
This mode indicates that the visual update should occur only when the UI queue is completely empty (indicating no further changes are coming).

VUM_DELAYED_VIA_APP_QUEUE

This mode indicates that the visual update should occur only when the application's queue is empty (indicating that no more changes are coming). This mode is very useful for making several changes that each require an update; it will put the updating off until the very end and will cause only a single update.

A **VisUpdateMode** is passed as a parameter to certain Vis and Gen messages. These messages typically have some effect on the visual representation of the objects—either on the geometry or on the image itself.

23.4.2.3 The Initial GState

MSG_VIS_VUP_CREATE_GSTATE

For any drawing to occur, a graphic state must exist. The graphic state typically is associated with the GenView's window and is created automatically, either when the window is first realized on the screen or when an exposure event occurs.

To create a GState, the system generates a MSG_VIS_VUP_CREATE_GSTATE and sends it to the object that first requires a visual update. This message travels up the visible tree until it gets to a window group (an object with its VTF_IS_WINDOW flag set), and then it creates a GState associated with that window object. Sometimes a content object will subclass this message to set up its own display information before the GenView window can create the new GState. The other time in which this message may be subclassed is by a content that manages a 32-bit graphics space; the content will have to translate the gstate to the proper position in the large document before letting other objects use it.

■ MSG_VIS_VUP_CREATE_GSTATE

```
GStateHandle MSG_VIS_VUP_CREATE_GSTATE();
```

This message travels up the visible object tree until it reaches either the root object or a window group object. It then creates a graphic state associated with that object; this graphic state is then used for the subsequent visual update. You may wish to subclass this message in order to alter the GState created or to set it up to take 32-bit coordinates. If you subclass it, however, be sure to call the superclass somewhere in the handler.

Source: Unrestricted—typically sent by a visible object to itself.

Destination: Any visible object—typically sent by a visible object to itself.

Parameters: None.

Return: The GStateHandle of the newly-created GState.

Interception: Unlikely—VisContent objects may intercept this message to translate the returned GState in a 32-bit graphics space. In this case, the VisContent must first call its superclass, then translate the GState appropriately before returning.

23.4.2.4 Retrieving the Current Window Handle

`MSG_VIS_QUERY_WINDOW`

Often a visible object will require the window handle of the window it's currently residing in. Some objects, for example, will want to force visual updates in real-time, and they can do this by using the current window handle, creating a special GState for it, and drawing to that GState. The visible object can either cache the window handle when the window is first opened, or it can retrieve it with MSG_VIS_QUERY_WINDOW.

■ MSG_VIS_QUERY_WINDOW

```
WindowHandle MSG_VIS_QUERY_WINDOW();
```

This message returns the window handle of the window the object currently resides in. In many cases, this will be the window handle of a GenView window.

Source: Unrestricted.

Destination: Any visible object—typically sent by a visible object to itself to retrieve its current window handle.

Parameters: None.

Return: The window handle of the object's current window.

Interception: Unlikely.

See Also: MSG_GEN_VIEW_GET_WINDOW.

23.4.2.5 Causing Redrawing

MSG_VIS_BOUNDS_CHANGED, MSG_VIS_REDRAW_ENTIRE_OBJECT, MSG_VIS_INVALIDATE, MSG_VIS_MARK_INVALID, MSG_VIS_INVAL_TREE, MSG_VIS_VUP_ADD_RECT_TO_UPDATE_REGION

If your visible tree contains a composite that manages its own children, chances are you'll need to force a visual update after certain changes occur. For example, if you have an object that can be moved with the mouse, it will probably want to erase its image and redraw itself somewhere else after the move; since the geometry manager is not involved here, the object must force a redrawing of the affected portion of the visible tree.

There are several ways an object can force a redraw. These all take the form of messages, each of which is described below.

MSG_VIS_DRAW

This message causes the entire visible branch to draw itself. This message may be sent by any object to any other object, including itself. It does not, however, update the screen image; you must call MSG_VIS_VUP_UPDATE_WIN_GROUP.

MSG_VIS_BOUNDS_CHANGED

This message causes the visible object to mark the affected portions invalid, causing the visual update mechanism to redraw that portion during the next visual update. This message is part of the visible update mechanism.

MSG_VIS_REDRAW_ENTIRE_OBJECT

This message causes the visible object to redraw itself entirely (as opposed to just the invalid portion) if it is currently drawable. Essentially, the object will send itself a MSG_VIS_DRAW when it receives this message. This is especially useful if an object needs to completely change its image.

MSG_VIS_INVALIDATE

This message invalidates the entire region inside the visible object's bounds. It will cause the window system to generate a MSG_META_EXPOSED for the affected area, causing a MSG_VIS_DRAW to propagate down the tree to all affected objects. Note that this message must be subclassed to work with scaled windows or large documents.

MSG_VIS_INVAL_TREE
This message invalidates the entire region inside the visible object's bounds, including any child window areas. The window system will generate a MSG_META_EXPOSED for all affected windows.

MSG_VIS_MARK_INVALID
This message allows the caller to set an object's invalid flags so the object will be properly updated later. The object is marked as being invalid, and the default handler ensures a visual update path is created. This method may *not* be subclassed by applications.

MSG_VIS_VUP_ADD_RECT_TO_UPDATE_REGION
This message adds the passed rectangular region to a window's update list. Essentially, the window group associated with the recipient object will ensure that the given rectangle gets marked invalid and included in the next visual update.

■ MSG_VIS_DRAW

```
void    MSG_VIS_DRAW(
        DrawFlags           drawFlags,
        GStateHandle        gstate);
```

This message causes the visible object to draw itself. The default behavior of this message for composite objects is to pass the message on to all children. There is no default behavior for **VisClass** for this message.

Source: Unrestricted.

Destination: Any visible object.

Parameters: *drawFlags* A record of type **DrawFlags** (defined below). The flags in this record describe what type of action initiated the MSG_VIS_DRAW.

gstate The GStateHandle of the gstate created for this visual update. All drawing done in the handler should be for this GState.

Return: Nothing.

Interception: All visible objects that wish to draw anything to the screen must intercept this message. Composites that intercept it must be sure to send it to their children (with **@send @visChildren**).

Structures: The flags in the **DrawFlags** record are listed below:

DF_EXPOSED This flag indicates the MSG_VIS_DRAW is a result of a MSG_META_EXPOSED from the window system. This is the standard and most usual form of MSG_VIS_DRAW.

DF_OBJECT_SPECIFIC
This flag is used by various Specific UI objects for object-specific things. Your visible objects should not be concerned with this flag.

DF_PRINT This flag indicates that the MSG_VIS_DRAW is a result of a MSG_META_EXPOSED_FOR_PRINT, the printing message. The GState passed will be a spooler GState rather than a window GState. If this flag is set, DF_EXPOSED will also be set.

DF_DONT_DRAW_CHILDREN
This flag indicates that composite objects should draw themselves but should *not* propagate the drawing message on to their children (this is nonstandard behavior).

DF_DISPLAY_TYPE This is actually an enumerated type indicating the display type being used by the system right now. The display type enumeration is named **DisplayClass**. For review, the types are listed below:

DC_TEXT, DC_GRAY_1, DC_GRAY_2, DC_GRAY_4, DC_GRAY_8, DC_COLOR_2, DC_COLOR_4, DC_CF_RGB.

■ MSG_VIS_BOUNDS_CHANGED

```
void     MSG_VIS_BOUNDS_CHANGED(@stack
         word   bottom,
         word   right,
         word   top,
         word   left);
```

The geometry manager sends this message when a geometry update changes an object's bounds.

Source: Unrestricted—sent by the geometry manager when a visible object's bounds have changed as the result of a geometry change.

Destination: Any visible object.

Parameters: *bottom, right, top, left*
The four parameters correspond to the rectangle defining the object's old bounds. The parameters are passed on the stack.

Return: Nothing.

Interception: Unlikely. If the object subclasses this message, it should invalidate any part of its old bounds that it might have drawn in (the default handler invalidates the entire passed range).

■ MSG_VIS_REDRAW_ENTIRE_OBJECT

```
void        MSG_VIS_REDRAW_ENTIRE_OBJECT();
```

This message causes the object send itself a MSG_VIS_DRAW, creating and destroying a GState for itself.

Source: Unrestricted—typically sent by an object to itself because of a state change, not as part of the visual update mechanism.

Destination: Any visible object—typically sent by an object to itself.

Interception: Unlikely.

■ MSG_VIS_INVALIDATE

```
void        MSG_VIS_INVALIDATE();
```

This message invalidates the entire region within the object's bounds. The message will cause the window system to generate a MSG_META_EXPOSED for the area covered by the bounds, causing a visual update to occur.

Source: Unrestricted.

Destination: Any visible object.

Interception: Must be intercepted if the object is working in a 32-bit document space. Must also be intercepted to work with scaled views.

■ MSG_VIS_INVAL_TREE

```
void        MSG_VIS_INVAL_TREE();
```

This message has the effect of MSG_VIS_INVALIDATE on an entire branch of the visible tree. The window system will generate MSG_META_EXPOSED for each affected window.

Source: Unrestricted.

Destination: Any visible object.

Interception: Unlikely.

■ MSG_VIS_MARK_INVALID

```
void        MSG_VIS_MARK_INVALID(
            VisOptFlags         flagsToSet,
            VisUpdateMode       updateMode);
```

This message allows the caller to set the recipient's *VI_optFlags* record so the object will get updated properly during the next visual update. The flags that can be set with this message are VOF_GEOMETRY_INVALID, VOF_WINDOW_INVALID, and VOF_IMAGE_INVALID. For more information on these flags, see section 23.3.4 on page 1306.

Source: Unrestricted.

Destination: Any visible object.

Parameters: *flagsToSet* A record of **VisOptFlags** indicating which type of invalidation is being caused. The flags allowed with this message are VOF_GEOMETRY_INVALID, VOF_WINDOW_INVALID, and VOF_IMAGE_INVALID.

updateMode A **VisUpdateMode** indicating when the visual update caused by this message should occur.

Return: Nothing.

Interception: *May not* be subclassed. Certain optimizations are made in the default handler, and subclassing may have unpredictable results.

■ MSG_VIS_ADD_RECT_TO_UPDATE_REGION

```
void        MSG_VIS_ADD_RECT_TO_UPDATE_REGION(@stack
            byte                unused,/* for word alignment of parameters */
            VisAddRectFlags     addRectFlags,
            word                bottom,
            word                right,
            word                top,
            word                left);
```

This message adds the passed rectangular region to the window group's list of regions that require visual update. The handler for this message will ensure that the window group object for this branch of the tree marks the region as invalid so it will be included in the next visual update. The *addRectFlags* parameter contains either of the flags listed below.

Source: Unrestricted—usually sent by an object to itself.

Destination: Any visible object—usually sent by an object to itself.

Parameters: *unused* An unused byte for alignment of parameters.

addRectFlags A record of **VisAddRectFlags**, defined below.

rectangle coords The coordinates of the rectangle to be added to the invalidation region.

Return: Nothing.

Interception: Unlikely—a composite object may wish to optimize invalidation by altering the passed bounds and then passing the message on to its superclass.

Structures: The **VisAddRectFlags** are listed below:

VARF_NOT_IF_ALREADY_INVALID
This flag indicates that the passed rectangle should not be invalidated if any visible object in the hierarchy going up to the window group object is marked invalid.

VARF_ONLY_REDRAW_MARGINS
This flag indicates that the object is invalidating old bounds and may indicate that the system can use a special optimization for invalidation.

23.4.2.6 Updating Window Groups

```
MSG_VIS_VUP_UPDATE_WIN_GROUP, MSG_VIS_UPDATE_WIN_GROUP,
MSG_VIS_UPDATE_WINDOWS_AND_IMAGE
```

A window group is an object that has window in which visible objects are displayed. Typically, you will not create your own window group objects, but you may need to notify a window group when it needs to be updated visually. To be a window group, an object has to have its VTF_IS_WINDOW or VTF_IS_PORTAL flag set in its *VI_typeFlags* field.

The window group is responsible for keeping track of the regions of its coordinate space that require visual updating. When a region or object gets invalidated, the window group remembers that it is invalid and makes sure it gets updated during the next visual update.

Any object can request that a visual update occur for the entire branch managed by a window group. It can use the three messages described in this section to request or force a visual update for the window group.

■ MSG_VIS_VUP_UPDATE_WIN_GROUP

```
Boolean  MSG_VIS_VUP_UPDATE_WIN_GROUP(
         VisUpdateMode updateMode);
```

This message travels up the visible tree from the recipient object to the first window group encountered; it will then cause that window group to go through a visible update by sending the window group object a MSG_VIS_UPDATE_WIN_GROUP, below.

Source: Unrestricted.

Destination: Any visible object residing in the window group that is to be updated.

Parameters: *updateMode* A **VisUpdateMode** indicating when the visual update should occur. VUM_MANUAL has the effect of a “no operation” because VUM_MANUAL does not cause visual updates.

Return: *True* if the update mechanism was invoked; *false* if i was not.

Interception: Not allowed.

Warnings: You may not subclass this message.

■ MSG_VIS_UPDATE_WIN_GROUP

```
void     MSG_VIS_UPDATE_WIN_GROUP(
         VisUpdateMode updateMode);
```

This message may only be sent by a window group object to itself. It causes the window group to actually go through a visual update.

Source: Visible update mechanism.

Destination: The window group object that is to be updated.

Parameters: *updateMode* A **VisUpdateMode** indicating when the visual update should occur. This is the same as passed to MSG_VIS_VUP_UPDATE_WIN_GROUP.

Return: Nothing.

Interception: Not allowed.

Warnings: You may not subclass this message.

■ MSG_VIS_UPDATE_WINDOWS_AND_IMAGE

```
void        MSG_VIS_UPDATE_WINDOWS_AND_IMAGE(
            VisUpdateImageFlags updateImageFlags);
```

This message is called by a window group during a visual update. It should not be sent or handled by anything other than a window group. It causes both the geometry and image of the affected branch of the visible tree to be updated, and it is used on branches which are already visually realized on the screen.

Source: Visual update mechanism.

Destination: A window group object.

Parameters: *updateImageFlags*

This is a record of **VisUpdateImageFlags** that govern the visual update of the window group.

Return: Nothing.

Interception: Not allowed.

Structures: The flags of **VisUpdateImageFlags** are listed below:

VUIF_ALREADY_INVALIDATED

An optimization flag to keep an already-invalid composite from invalidating all its children.

VUIF_SEND_TO_ALL_CHILDREN

A flag to ensure that all children get invalidated; it is used when the composite only updates its margins for optimization.

Warnings: You may not subclass this message.

23.4.2.7 Using Visual Monikers

```
MSG_VIS_DRAW_MONIKER, MSG_VIS_GET_MONIKER_POS,
MSG_VIS_GET_MONIKER_SIZE, MSG_VIS_FIND_MONIKER,
MSG_VIS_CREATE_VIS_MONIKER
```

Although visual monikers are typically used with generic objects (and are, in fact, documented in the **GenClass** chapter), you can display monikers with visible objects as well. The four messages discussed in this section can be used to draw a visible moniker, get information about a moniker, or locate a particular moniker in the visible object tree.

■ MSG_VIS_DRAW_MONIKER

```
void    MSG_VIS_DRAW_MONIKER(@stack
        DrawMonikerFlags    monikerFlags,
        ChunkHandle         visMoniker,
        word                textHeight,
        GStateHandle        gstate,
        word                yMaximum,
        word                xMaximum,
        word                yInset,
        word                xInset);
```

This message draws a visual moniker for the object. This message may be called by an object on itself in its MSG_VIS_DRAW handler.

Source: Unrestricted.

Destination: Any visible object.

Parameters: *monikerFlags* A record of **DrawMonikerFlags** indicating how the moniker should be drawn. These flags are described below.

visMoniker The chunk handle of the chunk containing the actual moniker to be drawn. This chunk must be in the same block as the object handling the message.

textHeight A parameter used for optimization—if the caller knows the system text height, it should pass it here. If not, it should pass zero.

gstate The handle of the GState to use when drawing the moniker. Typically, this is received by MSG_VIS_DRAW, which invokes this message.

yMaximum The maximum bottom bound of the moniker if DMF_CLIP_TO_MAX_WIDTH is set in *monikerFlags*. Pass MAX_COORD to avoid clipping.

xMaximum The maximum right bound of the moniker if DMF_CLIP_TO_MAX_WIDTH is set in *monikerFlags*. Pass MAX_COORD to avoid clipping.

yInset The vertical inset at which to begin drawing the moniker, if left or right justified.

xInset The horizontal inset at which to begin drawing the moniker, if top or bottom justified.

Return: Nothing.

Interception: Unlikely.

Structures: The **DrawMonikerFlags** structure is defined as follows:

```
typedef ByteFlags DrawMonikerFlags;
#define DMF_UNDERLINE_ACCELERATOR   0x40
            /* Underlines accelerator key, if any */
#define DMF_CLIP_TO_MAX_WIDTH       0x20
            /* Clips the moniker to the xMaximum
             * parameter */
#define DMF_NONE                    0x10
            /* Set to draw the moniker at the
             * current pen position (ignore the
             * xInset and yInset parameters) */
#define DMF_Y_JUST_MASK             0x0c
#define DMF_X_JUST_MASK             0x03
            /* These are two bitfields that
             * determine the justification.
             * Their offsets are below; they
             * are of type Justification. */
#define DMF_Y_JUST_OFFSET     2
#define DMF_X_JUST_OFFSET     0
```

■ MSG_VIS_GET_MONIKER_POS

```
XYValueAsDWord MSG_VIS_GET_MONIKER_POS(@stack
         DrawMonikerFlags    monikerFlags,
         ChunkHandle         visMoniker,
         word                textHeight,
         GStateHandle        gstate,
         word                yMaximum,
         word                xMaximum,
         word                yInset,
         word                xInset);
```

This message returns the position at which the moniker would appear if it were drawn with MSG_VIS_DRAW_MONIKER. The moniker is not actually drawn by this message.

Source: Unrestricted.

Destination: Any visible object.

Parameters: See MSG_VIS_DRAW_MONIKER above.

Return: A dword value representing the horizontal and vertical positions where the moniker would be drawn if it were drawn with the passed parameters. The horizontal position is returned in the high word; the vertical position is returned in the low word. Use the DWORD_X and DWORD_Y macros to extract the *x* and *y* values. These macros can be found in the file **graphics.h**.

Interception: Unlikely.

■ MSG_VIS_GET_MONIKER_SIZE

```
SizeAsDWord MSG_VIS_GET_MONIKER_SIZE(@stack
          DrawMonikerFlags    monikerFlags,
          ChunkHandle         visMoniker,
          word                textHeight,
          GStateHandle        gstate,
          word                yMaximum,
          word                xMaximum,
          word                yInset,
          word                xInset);
```

This message returns the size of the moniker specified by the parameters. The moniker is not drawn.

Source: Unrestricted.

Destination: Any visible object.

Parameters: See MSG_VIS_DRAW_MONIKER above.

Return: A dword value representing the size of the moniker. The width of the moniker is returned in the high word; the height is returned in the low word. Use the macros DWORD_WIDTH and DWORD_HEIGHT, which can be found in the file **visC.goh** and also on page 1336.

Interception: Unlikely.

■ MSG_VIS_FIND_MONIKER

```
optr      MSG_VIS_FIND_MONIKER(@stack
          VisMonikerSearchFlags  searchFlags,
          MemHandle              destBlock,
          ChunkHandle            monikerList,
          DisplayType            displayType);
```

This message locates the given visual moniker list and returns the optr of the moniker most appropriate for the passed display scheme.

Source: Unrestricted.

Destination: Any visible object.

Parameters: *searchFlags* A record of **VisMonikerSearchFlags** (described below) indicating the attributes of the moniker that should be returned.

destBlock Handle of the block into which the moniker chunk should be copied.

monikerList Chunk handle of the chunk containing the moniker list to search. The moniker list must be in the same object block as the object handling the message.

displayType The **DisplayType** of the moniker to be found (see below).

Interception: Unlikely.

Structures: The **VisMonikerSearchFlags** are listed below.

```
typedef WordFlags VisMonikerSearchFlags;
#define VMSF_STYLE            0xf000
      /* Four bits defining the preferred style of
       * the moniker. These bits are of type
       * VMStyle, which is defined below. */
#define VMSF_COPY_CHUNK       0x0400
      /* Set if the moniker should be copied into
       * the block specified if not already in
       * that block. */
#define VMSF_REPLACE_LIST     0x0200
      /* Set if the moniker list chunk containing
       * the VisMoniker should be replaced. The
       * chunk handle of the list will then point
       * to the moniker rather than the list. */
#define VMSF_GSTRING          0x0100
      /* Set if a GString moniker is expected,
       * clear if a text moniker is expected. */
      /* The remaining bits of this record are
       * reserved for internal use. */
#define VMSF_STYLE_OFFSET 12
typedef ByteEnum VMStyle;
#define VMS_TEXT              0
                  /* Normal text moniker */
#define VMS_ABBREV_TEXT       1
                  /* short text abbreviation */
#define VMS_GRAPHIC_TEXT      2
                  /* textual GString */
#define VMS_ICON              3
                  /* normal GString moniker */
#define VMS_TOOL              4
                  /* tool moniker, normally smaller
                   * than a standard moniker */
```

The **DisplayType** flags are listed below and can be found in win.h:

```
typedef ByteFlags DisplayType;
#define DT_DISP_SIZE                0xc0
            /* Two bits indicating the size of the
             * display; a DisplaySize value, one of
             * DS_TINY (CGA, or 256 x 320),
             * DS_STANDARD (EGA, VGA, HGC, MCGA),
             * DS_LARGE (800 x 600 SVGA), or
             * DS_HUGE (huge screens). */
#define DT_DISP_ASPECT_RATIO        0x30
            /* Two bits indicating the aspect
             * ratio of the screen; a value of
             * DisplayAspectRatio, one of
             * DAR_NORMAL (VGA or MCGA),
             * DAR_SQUISHED (EGA or HGC), or
             * DAR_VERY_SQUISHED (CGA) */
#define DT_DISP_CLASS               0x0f
            /* Four bits indicating the class of
```

```
                    * the display driver (or closest
                    * match); A DisplayClass value, one of
                    * DC_TEXT (char only, not implemented),
                    * DC_GRAY_1 (1 bit/pixel gray scale),
                    * DC_GRAY_2 (2 bit/pixel gray scale),
                    * DC_GRAY_4 (4 bit/pixel gray scale),
                    * DC_GRAY_8 (8 bit/pixel gray scale),
                    * DC_COLOR_2 (2 bit/pixel color index),
                    * DC_COLOR_4 (4 bit/pixel color index),
                    * DC_COLOR_8 (8 bit/pixel color index),
                    * DC_CF_RGB (color with RGB values) */
```

■ MSG_VIS_CREATE_VIS_MONIKER

```
ChunkHandle MSG_VIS_CREATE_VIS_MONIKER(@stack
          CreateVisMonikerFlags   flags,
          word                    height,
          word                    width
          word                    length,
          VisMonikerDataType      dataType,
          VisMonikerSourceType    sourceType,
          dword                   source);
```

This message creates a new chunk for a visual moniker within the recipient's object block. The new moniker can be created from an already existing visual moniker, a visual moniker list, a text string, a GString, or a token from the token database. The source may be defined by a far pointer, a global memory handle, or an optr. If the source is a text string or GString, a visual moniker structure will be created for the string. The newly-created chunk is marked dirty if the CVMF_DIRTY flag is passed.

If a moniker list is passed, the entire list will be copied into the destination object block. You must make sure that all the monikers in the list will still exist when the list is used.

Source: Unrestricted.

Destination: Any visible object.

Parameters: *flags* A record of **CreateVisMonikerFlags**. Currently only one may be passed: CVMF_DIRTY, which indicates that the chunk should be marked dirty.

height The height of the moniker.

width The width of the moniker.

length The length of the moniker, if a text string.

dataType The type of moniker, an enumeration of the type **VisMonikerDataType**. This parameter determines the type of moniker to be created.

sourceType The type of source used to create the new moniker, an enumeration of **VisMonikerSourceType**.

source This parameter can be a pointer to the source, the global handle of the block containing the source, or an optr pointing to the chunk containing the source, depending on the value in *sourceType*.

Return: The chunk handle of the new visual moniker. The visual moniker chunk resides in the object block of the object receiving the message. If the flag CVMF_DIRTY is passed, the chunk will be marked dirty.

Interception: Unlikely—custom UI gadgets may intercept in some cases.

Structures: **VisMonikerSourceType** and **VisMonikerDataType** are defined below. Both can be found in **visC.goh**.

```
typedef ByteEnum VisMonikerSourceType;
#define VMST_FPTR             0
            /* Source is referenced by a pointer.
             * CVMF_source is a far pointer. */
#define VMST_OPTR             1
            /* Source is referenced by an optr.
             * CVMF_source is an optr. */
#define VMST_HPTR             2
            /* Source is referenced by a combination
             * memory handle and offset into the
             * memory block (as opposed to an optr
             * in which the low word is actually a
             * chunk handle, not an offset). */

typedef ByteEnum VisMonikerDataType;
#define VMDT_NULL             0
            /* There is no source. Not valid for
             * MSG_VIS_CREATE_VIS_MONIKER. */
#define VMDT_VIS_MONIKER      1
            /* Source is a complete VisMoniker
             * structure. CVMF_length indicates the
             * size of a complete VisMoniker
             * structure; CVMF_width and
             * CVMF_height are unused. */
#define VMDT_TEXT             2
            /* Source is a text string. If the
             * string is null-terminated,
             * CVMF_length should be zero.
             * Otherwise, CVMF_length is the length
             * of the string; A VisMoniker
             * structure will be created for the
             * string. CVMF_width and CVMF_height
             * are unused. */
#define VMDT_GSTRING          3
            /* Source is a GString. If CVMF_length
             * is zero, the GString length is
             * determined by scanning for
             * GR_END_STRING. Otherwise, CVMF_length
             * indicates the length of the GString.
             * CVMF_width and CVMF_height indicate
             * the width and height of the GString.
             * If either is zero, the dimension will
             * be calculated by examining the
             * string. A VisMoniker structure will
             * be created for the GString. */
#define VMDT_TOKEN            4
            /* Source is a GeodeToken. CVMF_length,
             * CVMF_width, and CVMF_height are
             * unused. The destination must be able
             * to use this data type because the
             * specific UI must decide which moniker
             * to choose from the token in the token
             * database. */
```

■ DWORD_HEIGHT

```
word     DWORD_HEIGHT(val);
         dword  val;
```

This macro extracts the height from a **SizeAsDWord** structure (dword).

Objects

■ DWORD_WIDTH

```
word      DWORD_WIDTH(val);
          dword  val;
```

This macro extracts the width from a **SizeAsDWord** structure (dword).

■ MAKE_SIZE_DWORD

```
SizeAsDWord MAKE_SIZE_DWORD(width, height);
          word   width, height;
```

This macro takes a width and height and creates a **SizeAsDWord** value.

23.4.2.8 Cached GStates

```
MSG_VIS_CREATE_CACHED_GSTATES,
MSG_VIS_RECREATE_CACHED_GSTATES,
MSG_VIS_DESTROY_CACHED_GSTATES
```

Many complex visible objects (such as VisText and VisSpline) have both a cached GState and a reference count. When the reference count goes from zero to one, the Vis object creates and caches a GState; when the reference count goes from one to zero, the cached GState is destroyed. **VisClass** has three messages to create, update, and destroy the cached GStates. None of these messages has any default behavior; they are provided for complex objects to handle should they need them.

■ MSG_VIS_CREATE_CACHED_GSTATES

```
void      MSG_VIS_CREATE_CACHED_GSTATES();
```

This message may be used to create and cache a GState, typically to avoid having a complex Vis object update several times for several simple operations (such as pointer events).

Source: Unrestricted.

Destination: Any visible object.

Interception: Must be intercepted to have any effect; there is no default behavior.

■ MSG_VIS_RECREATE_CACHED_GSTATES

```
void        MSG_VIS_RECREATE_CACHED_GSTATES();
```

This message may be used to have the Vis object destroy and recreate any cached GStates it has. For example, cached GStates of a complex visible object that gets moved or resized will no longer map to the proper place in the object's document. After the move or resize, this message may be used to update those GStates.

23.4

Source: Unrestricted.

Destination: Any visible object.

Interception: Must be intercepted to have any effect; there is no default behavior.

■ MSG_VIS_DESTROY_CACHED_GSTATES

```
void        MSG_VIS_DESTROY_CACHED_GSTATES();
```

This message may be used to destroy any cached GStates the visible object may have.

Source: Unrestricted.

Destination: Any visible object.

Interception: Must be intercepted to have any effect; there is no default behavior.

23.4.3 Positioning Visible Objects

Every visible object has an instance data field called *VI_bounds* that records the object's visible bounds. Visible bounds are the left, top, right, and bottom bounds of a rectangle that define the maximum area covered by the visible object. The bounds are used primarily for determining the object's size and location.

The *VI_bounds* field is a **Rectangle** structure, provided by the graphics system. This structure has four components, as shown below:

```
typedef struct {
    sword   R_left;    /* x of upper-left corner */
    sword   R_top;     /* y of upper-left corner */
    sword   R_right;   /* x of lower-right corner */
    sword   R_bottom;  /* y of lower-right corner */
} Rectangle;
```

The bounds of non-window objects are stored in document coordinates relative to the document displayed by the window in which they appear. The left and top bounds constitute the position of the object. The difference between the right and left bounds is the object's width; the difference between bottom and top is its height. Together, the width and height constitute the object's size.

The bounds of a window or portal object (VTF_IS_WINDOW or VTF_IS_PORTAL) represent the location of the window, in document coordinates, within its parent window. The coordinate system within the window object is independent from that of its parent.

Unless an object's bounds are set in the Goc code, they will be set at zero and must be initialized to some values before the object can be drawn. This initialization can be done with the messages that set position and size, or it can be done automatically by the geometry manager (if a composite is managing its children).

The *VI_bounds* instance field exists in **VisClass** and therefore occurs also in **VisCompClass** and **VisContentClass** and all their subclasses. Any object can directly set any or all of its bounds; other objects (such as a parent VisComp) can direct the object to set its bounds to certain coordinates as a result of either geometry management or some user action. For example, a VisComp that has ten children may recalculate their positions if one of those children gets resized; the composite must then notify all the children of their new positions. To learn how a composite object can manage its children in different ways, see "VisComp," Chapter 24.

23.4.3.1 Object Position and Bounds

MSG_VIS_GET_POSITION, MSG_VIS_SET_POSITION, MSG_VIS_GET_BOUNDS

An object's bounds determine both the location and the size of the object in the document space. The object may reposition itself or resize itself at will; for example, it may allow the user to move or resize it with the mouse. The object can also receive direction from other objects that it should change its bounds. Note that none of these messages by themselves will cause the object to redraw; you must also send a MSG_VIS_MARK_INVALID to the object to get either its geometry or its image to be marked invalid.

An object's position is represented by the upper-left corner of its bounds, defined by *VI_bounds.R_top* and *VI_bounds.R_left*. You can retrieve an object's position by sending it MSG_VIS_GET_POSITION, or you can set it with MSG_VIS_SET_POSITION. You can also use MSG_VIS_GET_BOUNDS to get all four components of the object's bounds.

An object can access its own bounds field with a simple pointer. Since every message has *pself* as an implicit parameter (a pointer to the object's instance data structure), any object can look at or change its bounds directly. (See the example in Code Display 23-2 on page ◆ 1358.)

Using Swat, you can see a visible object's bounds with the command **pobj**.

If you have a composite object managing several children, you can arrange to have their positions determined automatically. This is discussed in section 23.4.3.3 on page 1343.

■ MSG_VIS_SET_POSITION

```
void      MSG_VIS_SET_POSITION(
          int    xOrigin,
          int    yOrigin);
```

This message causes the object to set its *VI_bounds* to the new location specified, retaining its current size. The left bound (*VI_bounds.R_left*) is set to the value of *xOrigin*, and the top bound (*VI_bounds.R_top*) is set to that of *yOrigin*. The right and bottom bounds are set to match the new origin and the width and height of the object.

Source: Unrestricted—usually the visible parent or geometry manager.

Destination: Any visible object.

Parameters:

xOrigin	New horizontal coordinate of the object's position.
yOrigin	New vertical coordinate of the object's position.

Return: Nothing.

Interception: Unlikely. It's much more likely that the object will subclass MSG_VIS_POSITION_BRANCH.

Warnings: This message will not cause an image invalidation. You must mark the object invalid with MSG_VIS_MARK_INVALID.

See Also: MSG_VIS_POSITION_BRANCH, MSG_VIS_MARK_INVALID.

■ MSG_VIS_GET_POSITION

```
XYValueAsDWord MSG_VIS_GET_POSITION();
```

This message returns the current origin of the object. The origin returned consists of the left and top object bounds (*VI_bounds.R_left* and *VI_bounds.R_top*, respectively).

Source: Unrestricted.

Destination: Any visible object.

Parameters: None.

Return: A dword value representing the left bound in the low word and the top bound in the high word. Use the macros DWORD_X and DWORD_Y to extract the bounds from the return value (see the file **graphics.h**).s

Interception: Unlikely.

■ MSG_VIS_GET_BOUNDS

```
void        MSG_VIS_GET_BOUNDS(
            Rectangle *retValue);
```

This message returns the current rectangle structure stored in the object's *VI_bounds* field. This rectangle represents both the position and the size of the object by giving its left, top, right, and bottom bounds.

Source: Unrestricted.

Destination: Any visible object.

Parameters: *retValue* A pointer to an empty **Rectangle** structure. The values in the object's *VI_bounds* field will be returned in the structure pointed to.

Return: The *retValue* pointer will be returned with the bounds filled into the **Rectangle** structure.

Interception: Unlikely.

23.4.3.2 Object Size

`MSG_VIS_GET_SIZE, MSG_VIS_SET_SIZE`

A visible object's size is defined as its width and height. The width is the difference between the object's left and right bounds; the height is the difference between the object's top and bottom bounds.

You can retrieve a visible object's size with the message MSG_VIS_GET_SIZE, and you can set it with MSG_VIS_SET_SIZE. MSG_VIS_SET_SIZE will set the object's right and bottom bounds without recalculating the object's position. If geometry is being managed automatically, you may want to subclass other messages; these are described in "Geometry Management" on page 1343.

In Swat, you can see the object's size indirectly with the command **pobj**. This command will show, among other things, the *VI_bounds* field of the object. You can manually calculate the size from that information.

■ MSG_VIS_SET_SIZE

```
void      MSG_VIS_SET_SIZE(
          int    width,
          int    height);
```

This message sets the object's height and width to the passed values, retaining the object's current position. The object's position will not be recalculated as a result of the size change.

Source: Unrestricted—usually the geometry manager or the object's visible parent calculating geometry.

Destination: Any visible object.

Parameters: *width* The new width of the object. *width* will be added to the object's left bound to get the object's new right bound.

height The new height of the object. *height* will be added to the object's top bound to get the object's new bottom bound.

Return: Nothing.

Interception: Unlikely.

Warnings: This message will not cause an image invalidation. You must mark the object invalid with MSG_VIS_MARK_INVALID.

■ MSG_VIS_GET_SIZE

```
SizeAsDWord MSG_VIS_GET_SIZE();
```

This message returns the current size (width and height) of the object. Both values are word-sized integers and are returned in a single dword value. The high word of this value is the object's width, and the low word is the object's height.

Source: Unrestricted.

Destination: Any visible object.

Parameters: None.

Return: A dword value representing the object's width in the low word and height in the high word. Use the macros DWORD_WIDTH and DWORD_HEIGHT to extract the width and height; see page 1336.

Interception: Unlikely.

23.4.3.3 Geometry Management

MSG_VIS_UPDATE_GEOMETRY, MSG_VIS_NOTIFY_GEOMETRY_VALID, MSG_VIS_RECALC_SIZE, MSG_VIS_GET_CENTER, MSG_VIS_RECALC_SIZE_AND_INVAL_IF_NEEDED, MSG_VIS_RESET_TO_INITIAL_SIZE, MSG_VIS_POSITION_BRANCH, MSG_VIS_POSITION_AND_INVAL_IF_NEEDED, MSG_VIS_INVAL_ALL_GEOMETRY

Geometry management consists of positioning and sizing visible objects properly. This can be done manually by the programmer or automatically by GEOS. If you wish to manually control your visible tree's geometry, set the flag VCGA_CUSTOM_MANAGE_CHILDREN in the top composite's *VCI_geoAttrs* field (see "VisComp," Chapter 24). You can then use any messages, either native to **VisClass** or defined by your subclasses, for object positioning and sizing.

Automatic geometry management occurs when the geometry of the visible tree is marked invalid somewhere and a visual update is initiated. The tree's geometry can be marked invalid in several ways—an object can be added to or removed from the tree, an object can be manually marked invalid (with MSG_VIS_MARK_INVALID), or an object can be opened or closed (MSG_VIS_OPEN or MSG_VIS_CLOSE).

When automatic geometry management is invoked, the geometry manager will recalculate the entire affected branch's geometry. It will calculate geometry for each affected window group, from the topmost affected window object down to the bottommost affected object. If the geometry manager encounters window groups below the topmost window, it will check first to see if the lower window's size is controlled by its children (e.g., a GenView that follows the geometry of its content). If the lower window is scrollable or otherwise not subject to its children's geometry, the geometry manager will finish calculating the geometry above the lower window before proceeding.

Calculating geometry is an involved, complex task. The process involves composites and their children negotiating on desired sizes. How the geometry is calculated depends primarily on the relationships between the objects and how their flags are set.

For example, if a GenView is set up to be exactly the same size as its VisContent object, the calculations will be different than if the GenView were scrollable. The same is true for visible composites that manage their children: If the composite resizes itself large enough to hold all its children, the children must be resized first.

The negotiation between parent and child in the visible tree takes the form of a single message: MSG_VIS_RECALC_SIZE. This is the single most important message in calculating the tree's geometry, though others are also significant. This message is sent by a composite to each of its children in turn; it passes a suggested size, and the child returns its desired size.

The composite collects all the desired sizes of its children and compares that to the size it thinks it should be. If necessary, it makes another pass through the children, or it passes on its desired size to its parent composite. Depending on the situation and the disparity between parent and children sizes, the geometry calculation may take a single pass or several. If the sizes can not be resolved through this negotiation, the geometry manager will make a decision after several passes, typically expanding the parent as much as possible to fit all the children.

After a visible object has calculated its size, its size will be set either by itself or by its parent. Often this is done with MSG_VIS_SET_SIZE. Once all sizes have been determined and set, the objects set their positions with MSG_VIS_POSITION_BRANCH, which propagates down the tree. It sets the position of the topmost composite and then sets each of its children's positions appropriately.

When geometry calculations are complete, the geometry manager will send a MSG_VIS_NOTIFY_GEOMETRY_VALID to all visible objects with the flag VGA_NOTIFY_GEOMETRY_VALID set in its *VI_geoAttrs* field.

Several other messages may be used during geometry updates. Some of them may be intercepted to alter the behavior of the default method. These messages are listed below. Others can be used to invalidate a visible object's geometry or to cause a geometry update on the tree.

MSG_VIS_UPDATE_GEOMETRY
This message is typically called by the UI during a window group update. It calculates the geometry of the entire branch receiving it. It's not likely you'll subclass this method—altering its behavior can be used for certain optimizations, however.

MSG_VIS_NOTIFY_GEOMETRY_VALID
This message will be sent to an object when all its geometry messages have been handled and its geometry has been fully resolved (at least for the moment). This message will be sent only if VGA_NOTIFY_GEOMETRY_VALID is set in the object's *VI_geoAttrs* field. There is no default behavior for this message; it should be intercepted only if the object needs to do calculations after the geometry has settled down.

MSG_VIS_RECALC_SIZE
This message is normally sent by the geometry manager to a composite object. The composite is expected to send it to all its children; the children will then determine their sizes, and the composite will figure the total size and return. A composite that manages its own children can also use this message to query its children for their desired sizes.

MSG_VIS_RECALC_SIZE_AND_INVAL_IF_NEEDED
This message is an optimized MSG_VIS_RECALC_SIZE. Composites should use this version when sending a recalculation message to their children. This message only causes recalculation if needed and will invalidate the object if the size gets recalculated. It invalidates the object's image at its old bounds and only invalidates its geometry if the bounds change. It is highly unlikely that you'll subclass this method.

MSG_VIS_GET_CENTER
This message returns the amount of space on each side the object wants reserved for itself. It does not, as the name suggests, return the coordinates of the object's center. Instead, it returns the distance in each direction the object requires (you may think of it as external margins).

MSG_VIS_RESET_TO_INITIAL_SIZE
This message propagates down the visible tree and causes every object that receives it to reset itself to its original size. It does this by clearing VGA_GEOMETRY_CALCULATED in the object's *VI_geoAttrs* field, causing the object to believe it is being put on-screen for the first time. This message will also set the object's flags to indicate that its geometry has become invalid. It's highly unlikely your application will subclass this method.

MSG_VIS_POSITION_BRANCH
This message is sent by the geometry manager when an entire visible branch should be repositioned. If an application wants to do something special, it should subclass the handler for this message. Typically, though, only subclasses of **VisCompClass** and perhaps **VisContentClass** will change this method.

MSG_VIS_POSITION_AND_INVAL_IF_NEEDED
This message is an optimized MSG_VIS_POSITION_BRANCH that causes a branch reposition only if it's necessary. It will also invalidate both the object's and the parent's original images and, if the position changes, the geometry as well. It is highly unlikely that you'll subclass this method.

MSG_VIS_INVAL_ALL_GEOMETRY
This message invalidates all geometry under the recipient object, forcing invalidation of all children. This is a brute-force message used only when absolutely necessary.

■ MSG_VIS_UPDATE_GEOMETRY

```
void	MSG_VIS_UPDATE_GEOMETRY();
```

This message is sent during a visual update to cause objects to recalculate their geometry. Applications can use this message to ensure that geometry gets updated for a visual branch even if the branch is not currently drawn on the screen.

Source: Visual update mechanism.

◆Objects

Destination: Any visible object.

Interception: May be intercepted for optimizations. A thorough knowledge of the geometry update mechanism and the specifics of the visible tree's geometry is strongly recommended, however.

■ MSG_VIS_NOTIFY_GEOMETRY_VALID

```
void      MSG_VIS_NOTIFY_GEOMETRY_VALID();
```

This message is sent by the geometry manager to objects that have VGA_NOTIFY_GEOMETRY_VALID set. The message is sent only after all geometry messages have been handled for the object. It has no default behavior.

Source: Visible update mechanism.

Destination: Any visible object—typically a child of the object sending the message.

Interception: This message has no default behavior. It should be intercepted by any object that needs to do something after its geometry has been made valid.

■ MSG_VIS_RECALC_SIZE

```
SizeAsDWord MSG_VIS_RECALC_SIZE(
          int    width,
          int    height);
```

This message takes the suggested height and width of the object and recalculates the object's desired size based on them. Composite objects are expected to pass this message to their children and then calculate their sizes based on those of their children. The default behavior of the handler for this is to return the current size no matter what the suggested size is.

Source: Visible update mechanism.

Destination: Any visible object.

Parameters:

width	Suggested new width of the object as determined by the object's parent.
height	Suggested new height of the object as determined by the object's parent.

Return: A dword value representing the object's desired size as calculated by the object. The high word of the return value represents the width and the low word the height. Use the macros DWORD_WIDTH and DWORD_HEIGHT to extract the width and height; see page 1336.

Interception: Default behavior is to return the object's current size no matter what the passed width and height are. Any object that wants special sizing behavior based either on the object's children or on the suggested size should subclass the method.

■ MSG_VIS_RECALC_SIZE_AND_INVAL_IF_NEEDED

```
SizeAsDWord MSG_VIS_RECALC_SIZE_AND_INVAL_IF_NEEDED(
            int    width,
            int    height);
```

This message is the same as MSG_VIS_CALC_RESIZE except that it is somewhat optimized. It will recalculate only if certain flags are set for the object, and it will mark the geometry invalid only if recalculation is done.

Source: Visible update mechanism.

Destination: Any visible object.

Parameters:

width	Suggested new width of the object as determined by the object's parent.
height	Suggested new height of the object as determined by the object's parent.

Return: A dword value representing the object's desired size as calculated by the object. The low word of the return value represents the width and the high word the height.

Interception: Unlikely—Default behavior is to return the object's current size no matter what the passed width and height are. Any object that wants special sizing behavior based either on the object's children or on the suggested size should subclass the method.

■ MSG_VIS_GET_CENTER

```
void        MSG_VIS_GET_CENTER(
            GetCenterParams *retValue);
```

This message returns the amount of space needed by the object in each direction from its center. It does not, as the name suggests, return the coordinates of the center of the object. Instead, the object must determine the amount of space it requires from its center to each edge for geometry management and return them in the structure passed.

Source: Visible update mechanism.

Destination: Any visible object

Parameters: *retValue* A pointer to an empty **GetCenterParams** structure, defined below.

Return: The method must fill the **GetCenterParams** structure pointed to by the *retValue* parameter.

Interception: Any object that wants to effect a different centering than the default should subclass this method. For example, if a set of objects can be overlapped, or if they should have extra space between them, they should subclass this message and alter the returned values appropriately.

Structures: The **GetCenterParams** structure is defined as follows:

```
typedef struct {
    word   GCP_aboveCenter;    /* space above */
    word   GCP_belowCenter;    /* space below */
    word   GCP_leftOfCenter;   /* space left */
    word   GCP_rightOfCenter;  /* space right */
} GetCenterParams;
```

■ MSG_VIS_RESET_TO_INITIAL_SIZE

```
void    MSG_VIS_RESET_TO_INITIAL_SIZE(
        VisUpdateMode updateMode);
```

This message propagates down an entire visual branch, causing the objects in it to reset their sizes to their original width and height. First it invalidates the object's geometry, then it recalculates the geometry starting at the top.

Source: Visible update mechanism.

Destination: Any visible object.

Parameters: *updateMode* A **VisUpdateMode** indicating when the visual update of the tree should occur.

Return: Nothing.

Interception: Unlikely.

■ MSG_VIS_POSITION_BRANCH

```
void    MSG_VIS_POSITION_BRANCH(
        word   xOrigin,
        word   yOrigin);
```

This message repositions an entire visible branch. It propagates down the branch, causing each visible object to reposition itself based on the new origins passed. Composites must pass the appropriate altered positions to their children.

Source: Visible update mechanism.

Destination: Any visible object.

Parameters: *xOrigin* The new horizontal position of the object relative to the document or window it's in. This value is set into the object's *VI_bounds.R_left* bound, and the right bound is set according to the object's size.

yOrigin The new vertical position of the object relative to the document or window it's in. This value is set into the object's *VI_bounds.R_top* bound, and the bottom bound is set according to the object's size.

Return: Nothing.

Interception: The default behavior is to set the object to the passed position. Composite objects that want to position their children a special way should subclass this (and *not* call the superclass) method. Non-composites will rarely, if ever, subclass this message.

■ MSG_VIS_POSITION_AND_INVAL_IF_NEEDED

```
Boolean   MSG_VIS_POSITION_AND_INVAL_IF_NEEDED(
          word   xPosition,
          word   yPosition);
```

This message is an optimized version of MSG_VIS_POSITION_BRANCH. It repositions the branch only if necessary and will mark the image invalid only if the position was changed. It returns a flag indicating whether or not the object was repositioned.

Source: Visible update mechanism.

Destination: Any visible object—typically sent by a composite to its children.

Parameters: *xPosition* The new horizontal position of the object relative to the document or window it's in.

yPosition The new vertical position of the object relative to the document or window it's in.

Return: If the object was repositioned, the message will return *true*. Otherwise, it will return *false*.

Interception: Unlikely.

■ MSG_VIS_INVAL_ALL_GEOMETRY

```
void        MSG_VIS_INVAL_ALL_GEOMETRY(
            VisUpdateMode       updateMode);
```

This is a brute-force, desperation message used to invalidate all geometry in the recipient's tree; that is, all geometry of the recipient and all objects under it in the tree will be invalidated. You should only use this message when the whole tree should be recalculated.

Source: Unrestricted.

Destination: The object requiring geometry recalculation.

Parameters: *updateMode* A visual update mode indicating when the visual update of the recalculation should occur.

Return: Nothing.

Interception: Generally not intercepted.

23.4.4 Handling Input

Input events, what they mean, and how they are generated are discussed fully in "Input," Chapter 11 of the Concepts Book. The visible world has some default behavior for handling input messages such as MSG_META_START_SELECT, MSG_META_END_SELECT, and MSG_META_PTR. For the most part, however, visible objects that want to respond to input must subclass and handle the input messages that interest them.

Composite objects will automatically pass mouse messages on to the proper child directly under the mouse pointer. Non-composite objects (direct subclasses of **VisClass**) have no special default behavior for input handling.

Note that **VisClass** does not intelligently handle layering of objects. That is, if one object is visibly on top of another object, it will not necessarily receive the input events—the events may be passed to the partially hidden object. This is due to the order in which drawing and mouse events are handled by a visible object's children.

Take the example of a visible composite with two children. Each child draws a filled rectangle covering its bounds; the first child has a width and height of 40, and the second child has a width and height of 20. The composite's declaration shows

```
    ...
    VCI_comp = LargeChild, SmallChild;
    ...
```

Because drawing occurs in the order the children are listed, the small child will be drawn over the large child, as in Figure 23-3. If each object can intercept and handle mouse events (e.g., MSG_META_START_SELECT), clicks will never reach the small object without special behavior added to the composite. This is because input events are handled in the same order as drawing; the input event will be given to the first child whose bounds include the event's coordinates.

So, a click event on (10, 10), which is in the middle of the small square, would first be passed to the LargeChild object. The large child would see that the click was within its bounds, and it would take the event. The event would never be handled by the small object.

Therefore, if you need to have overlapping objects which handle user input, you will have to add special features for bounds detection. In the example of the two squares, if neither were to move, the composite could simply pass the event to the small child first and then to the large child, assuming the small child is always on top. If the objects could be moved forward and back so they hide each other, the composite (or the objects themselves) will need some other, more complex, detection behavior.

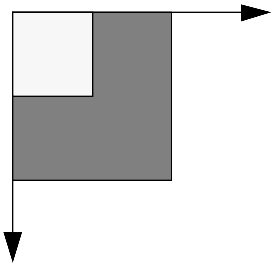


Figure 23-3 *Drawing Order of Vis Objects*
Since the large object is the first child, it is drawn first. The second child is drawn on top of it.

23.4.4.1 Mouse Events

MSG_META_PTR, MSG_META_START_SELECT, MSG_META_END_SELECT, MSG_META_DRAG_SELECT, MSG_META_DRAG, MSG_META_START_MOVE_COPY, MSG_META_DRAG_MOVE_COPY, MSG_META_START_OTHER, MSG_META_END_OTHER, MSG_META_DRAG_OTHER

See Also

Mouse events are referenced in "System Classes," Chapter 1, starting on

Composite objects will automatically pass mouse messages on to the first child directly under the mouse pointer. Non-composite objects (direct subclasses of **VisClass**) have no special default behavior for input handling.

Typically, a visible object that needs to handle mouse input will subclass MSG_META_START_SELECT (or one of the other press events such as MSG_META_START_MOVE_COPY). This message is sent when the user presses on the "select" mouse button while the pointer is over the object. In this method, the object should grab the mouse (as shown below). When it has the mouse grab, it will receive MSG_META_PTR each time the mouse is moved; the object should subclass this message to provide the desired reaction to mouse moves. Finally, when the user releases the select button, the object will receive a MSG_META_END_SELECT. At this point, it should release the mouse grab.

Typically, mouse events will arrive with coordinates specified in 16 bits. This is the default graphics space, and it represents a coordinate system more than 25 yards on a side. Nearly all applications will find this graphic system big enough for their uses. However, if you need larger coordinates, you can use the large document space with 32-bit coordinates. To do this, you must set the GVA_WINDOW_COORDINATE_MOUSE_EVENTS in the *GVI_attrs* field of your GenView object and set up your visible objects correctly. For more information on supporting 32-bit graphic spaces with visible objects, see "Visible Layers and 32-Bit Graphics" on page 1393.

Grabbing the Mouse

MSG_VIS_GRAB_MOUSE, MSG_VIS_FORCE_GRAB_MOUSE, MSG_VIS_GRAB_LARGE_MOUSE, MSG_VIS_FORCE_GRAB_LARGE_MOUSE, MSG_VIS_RELEASE_MOUSE

When an object "grabs the mouse," it requests that the UI send all future mouse events directly to that object. Normally, mouse events will travel down the visible object tree until they reach either the leaf object under the pointer or an object under the pointer that handles them. When an object has the mouse grab, the mouse events will go to the object with the grab regardless of the location of the pointer.

Typically, if an object wants to receive mouse events, it will subclass MSG_META_START_SELECT. This message is sent to the object when the pointer is over the object's bounds and the user presses the select button. In this handler, the object grabs the UI's gadget exclusive and then grabs the mouse. A good example of this can be found in the TicTac sample application's handler for MSG_META_START_SELECT.

After an object grabs the mouse, it receives all pointer and drag events that occur. These are primarily MSG_META_PTR messages, and the object must subclass MSG_META_PTR to achieve the proper behavior it wants. When the user lets go of the select button, the object will receive a MSG_META_END_SELECT. If it grabbed the mouse (which it probably did if it receives this message), it must relinquish both the gadget exclusive and its mouse grab.

An example of getting and releasing the mouse grab (taken from the TicTac sample application) is shown in Code Display 23-2. The object subclasses these messages to let the user drag it around the screen.

The UI allows the visible object to get the normal mouse grab by using four different messages. Each of these is detailed below. If you're using a large document (32-bit coordinates), you will use the "large" mouse grabs. Otherwise, the normal mouse grab messages will do.

■ MSG_VIS_GRAB_MOUSE

```
void    MSG_VIS_GRAB_MOUSE();
```

A visible object sends itself this message (with **@call**) when it wants to acquire the normal mouse grab. The message MSG_VIS_RELEASE_MOUSE must be used to relinquish the grab.

Source: Unrestricted.

Destination: Any visible object—typically sent by the object to itself.

Interception: Highly unlikely.

■ MSG_VIS_FORCE_GRAB_MOUSE

```
void        MSG_VIS_FORCE_GRAB_MOUSE();
```

A visible object sends itself this message (with **@call**) when it wants to forcibly grab the mouse input. The object will acquire the mouse grab even if another object currently has it; the other object will receive a MSG_VIS_LOST_GADGET_EXCL when it loses the mouse by a forced grab. The object must release the grab later with MSG_VIS_RELEASE_MOUSE.

Source: Unrestricted.

Destination: Any visible object—typically sent by the object to itself.

Interception: Highly unlikely.

■ MSG_VIS_GRAB_LARGE_MOUSE

```
void        MSG_VIS_GRAB_LARGE_MOUSE();
```

A visible object sends itself this message (with **@call**) when it wants to acquire the mouse grab and when it wants 32-bit coordinates. If the object wants 16-bit coordinates, it should use MSG_VIS_GRAB_MOUSE, above. The object must relinquish the mouse grab with MSG_VIS_RELEASE_MOUSE.

Source: Unrestricted.

Destination: Any visible object—typically sent by the object to itself.

Interception: Highly unlikely.

■ MSG_VIS_FORCE_GRAB_LARGE_MOUSE

```
void        MSG_VIS_FORCE_GRAB_LARGE_MOUSE();
```

A visible object sends itself this message (with **@call**) when it wants to forcibly grab the mouse input and when it wants large (32-bit) input coordinates. If it wants normal (16-bit) coordinates, it should use MSG_VIS_FORCE_GRAB_MOUSE, above. The object will acquire the mouse grab even if another object currently has it; the other object will receive a MSG_VIS_LOST_GADGET_EXCL when it loses the mouse by a forced grab. The object must release the grab later with MSG_VIS_RELEASE_MOUSE.

Source: Unrestricted.

Destination: Any visible object—typically sent by the object to itself.

Interception: Highly unlikely.

■ MSG_VIS_VUP_ALTER_INPUT_FLOW

```
void    MSG_VIS_VUP_ALTER_INPUT_FLOW(@stack
        PointDWord          translation,
        WindowHandle        window,
        optr                object,
        word                grapTypeAndFlags);
```

This message is the primitive employed by the grab/release messages shown above. Objects will rarely use this unless they are large objects (using 32-bit coordinates). Large objects and composites should subclass this message. In the case of a mouse grab, they should add in their 32-bit offset amounts to the translation structure before passing the event on to their superclasses. This will result in large mouse events being properly translated for the duration of the mouse grab.

Source: Unrestricted.

Destination: Any visible object—typically a VisContent which uses a 32-bit graphics space for its children.

Parameters:

translation The offsets to the origin of the “local” 16-bit graphics space.

window The handle of the view window associated with this input flow alteration.

object The optr of the object altering the input flow.

grabTypeAndFlags
The type and flags for the input flow grab affected by this alteration. The high byte contains a record of **VisInputFlowGrabType**, described below; the low byte contains a **VisInputFlowGrabFlags** record, also described below.

Return: Nothing.

Interception: Any VisContent object that uses a large graphics space should subclass this method. In the handler, it should apply the stated translation, then call its superclass. Each subsequent mouse event will have the translation applied before being passed on to the composite’s children.

Structures: The types of **VisInputFlowGrabType** are listed below:

VIFGT_ACTIVE The affected grab type is the active grab.

VIFGT_PRE_PASSIVE
The affected grab type is the pre-passive grab.

VIFGT_POST_PASSIVE
The affected grab type is the post-passive grab.

The flags of **VisInputFlowGrabFlags** are listed below:

VIFGF_NOT_HERE Flag set to indicate that the message should not be handled in this method but rather passed up the tree with this flag cleared. It is used to distinguish between requests made by other objects and requests made by an object sent to itself. It also allows an object to redirect the message to another node not in the visible hierarchy by handling the message and passing it in a direction other than up the visible tree. This overrides all the other flags in this record.

VIFGF_FORCE Flag set if the grab should be forced. Note that both VIFGF_GRAB and VIFGT_ACTIVE must also be specified.

VIFGF_GRAB Flag set if the grab is being made, clear if the grab is being released. In either case, *object* must be passed—if a release, the grab will not be released unless the objects match. Note also that only one object may have the active grab, but many objects may have either passive grab at a given time.

VIFGF_KBD Flag set if action is a keyboard grab or release.

VIFGF_MOUSE Flag set if action is a mouse grab or release.

VIFGF_LARGE Flag set if large mouse events are affected by the flow alteration. VIFGF_MOUSE must also be set.

VIFGF_PTR Flag set if mouse grab and pointer events are requested. If flag clear, only button-related events will be sent. VIFGF_MOUSE must also be set.

■ MSG_VIS_RELEASE_MOUSE

```
void      MSG_VIS_RELEASE_MOUSE();
```

A visible object sends itself this message if it has the mouse grab and wants to release it. Typically, this message will be called in the object's MSG_META_END_SELECT handler. It works for both large and small grabs.

Source: Any visible object.

Destination: Any visible object—typically sent by the object to itself.

Interception: Unlikely.

Code Display 23-2 Grabbing and Releasing the Mouse

```
/* This code display shows the MSG_META_START_SELECT, MSG_META_END_SELECT,
 * and MSG_VIS_LOST_GADGET_EXCL handlers from the TicTac sample application.
 *
 * The sequence of input events and the messages they generate follow the
 * following basic patterns:
 * 1.   User presses select button
 *      MSG_META_START_SELECT generated, sent to object under mouse pointer
 *      If object under pointer is a game piece,
 *              Game piece grabs gadget exclusive
 *              Game piece grabs mouse
 * 2.   User holds button and drags mouse
 *      MSG_META_DRAG_SELECT generated, sent to object with mouse grab
 *              Game piece sets internal "dragging" flag
 *              Game piece draws outline
 *      MSG_META_PTR messages generated during drag, sent to object with mouse grab
 *              Game piece erases previous outline
 *              Game piece draws new outline at new pointer position
 * 3.   User releases button
 *      MSG_META_END_SELECT generated, sent to object with mouse grab
 *              Game piece releases gadget exclusive (MSG_VIS_RELEASE_GADGET_EXCL)
 *              MSG_VIS_LOST_GADGET_EXCL generated, sent to mouse grab (game piece)
 *              Game piece erases old outline, if any
 *              Game piece moves itself, erases old bounds, draws new bounds
 *              Game piece releases mouse grab (MSG_VIS_RELEASE_MOUSE)
 */

/***********************************************************************
 *              MSG_META_START_SELECT for TicTacPieceClass
 ***********************************************************************
 * SYNOPSIS:    Grabs the mouse and calls for future pointer events.
 *              When the user clickes in the view, TicTacView will pass
 *              the click event to TicTacBoard. Since TicTacBoardClass
 *              does not intercept the event, VisContentClass passes
 *              it on to its child object currently under the pointer.
 * PARAMETERS:
 *      void    (MouseReturnParams *retVal,
 *               word xPosition, word yPosition, word inputState);
 *
 * STRATEGY:    When the piece object receives this message, it means
 *              it has been clicked on by the user and the mouse button
 *              is still down. The piece must grab the mouse so that it
 *              gets all future mouse events, and it must request that
 *              all future mouse events be sent to it. This ensures
 *              that if the pointer leaves the object's bounds while
 *              the button is still pressed, the piece object will still
 *              receive all the pointer events (otherwise they would be
 *              sent to whatever object was under the new pointer position).
 ***********************************************************************/

@method TicTacPieceClass, MSG_META_START_SELECT {

        /* Grab the gadget exclusive so we're allowed to grab the mouse. */
    @call @visParent::MSG_VIS_TAKE_GADGET_EXCL(oself);

        /* Grab the mouse. This requests that all future pointer
         * events be passed directly to this game piece. */
    @call self::MSG_VIS_GRAB_MOUSE();

        /* Finally, return that this particular click event has been processed.
         * If this flag is not returned, the system will send out the click
         * event again. */
    retVal->flags = MRF_PROCESSED;        /* this event processed */
}

/***********************************************************************
 *              MSG_META_END_SELECT for TicTacPieceClass
 ***********************************************************************
 * SYNOPSIS:    This message is received when the selection button has
 *              been released and this game piece had the mouse grab.
 *              All it does is release the gadget exclusive.
 * PARAMETERS:
 *      void    (MouseReturnParams *retVal,
 *               word xPosition, word yPosition, word inputState);
 ***********************************************************************/
```

```
@method TicTacPieceClass, MSG_META_END_SELECT {
    @call @visParent::MSG_VIS_RELEASE_GADGET_EXCL(oself);
    retVal->flags = MRF_PROCESSED;       /* this event processed */
}

/***********************************************************************
 *              MSG_VIS_LOST_GADGET_EXCL for TicTacPieceClass
 ***********************************************************************
 * SYNOPSIS:    This message is received when the selection button has
 *              been released and this game piece had the mouse grab.
 *              It first checks to see if the new, proposed bounds are
 *              on the game board. If the bounds are valid, then
 *              it sets the objects VI_bounds field to the new values
 *              and causes the object to erase its original drawing
 *              and draw itself at its new bounds. If the bounds are
 *              not on the game board, it will retain the original bounds
 *              and redraw using them.
 * PARAMETERS:  void ();
 ***********************************************************************/

@method TicTacPieceClass, MSG_VIS_LOST_GADGET_EXCL {
    WindowHandle win;            /* window handle of view window */
    GStateHandle gstate;         /* temporary gstate to draw to */

    /* See if piece was being dragged or not. If so, clear the outline. */

    if (pself->TTP_dragging) {

            /* Get the window handle of the view window and create a
             * temporary gstate for it to draw to. */

        win = @call TicTacView::MSG_GEN_VIEW_GET_WINDOW();
        gstate = GrCreateState(win);

            /* Set the line color and mode for drawing the outline. */

        GrSetLineColor(gstate, CF_INDEX, C_BLACK, 0, 0);
        GrSetMixMode(gstate, MM_INVERT);

            /* Erase outline on screen. */

            /* If the game piece type is TTPT_BOX, draw a rectangle
             * outline. Otherwise draw an ellipse outline. */

        if (pself->TTP_pieceType == TTPT_BOX) {
            GrDrawRect(gstate, pself->TTP_horizPos, pself->TTP_vertPos,
                         ((pself->TTP_horizPos) + PIECE_WIDTH),
                         ((pself->TTP_vertPos) + PIECE_HEIGHT));
        } else {
```

```
    GrDrawEllipse(gstate, pself->TTP_horizPos, pself->TTP_vertPos,
                 ((pself->TTP_horizPos) + PIECE_WIDTH),
                 ((pself->TTP_vertPos) + PIECE_HEIGHT));
}

    /* Check to see if the new bounds are on the game board. If
     * they are, set the object's bounds to the new values. If
     * the are not, retain the original values and set the values
     * to those last stored. */

if (@call TicTacBoard::MSG_TICTAC_VALIDATE_BOUNDS(
                ((pself->TTP_vertPos) + PIECE_HEIGHT),
                ((pself->TTP_horizPos) + PIECE_WIDTH),
                pself->TTP_vertPos, pself->TTP_horizPos)) {

    /* Invalidate the original drawing of the game piece. Send
     * the VI_bounds rectangle as the parameters because they have
     * not been changed since the START_SELECT. This message is
     * the equivalent of calling GrInvalRect() with the same bounds */

    @call self::MSG_VIS_BOUNDS_CHANGED(pself->VI_bounds.R_bottom,
                             pself->VI_bounds.R_right,
                             pself->VI_bounds.R_top,
                             pself->VI_bounds.R_left);

    /* Set the game piece object's bounds to the new coordinates. */

    pself->VI_bounds.R_left = pself->TTP_horizPos;
    pself->VI_bounds.R_right = (pself->TTP_horizPos) + PIECE_WIDTH;
    pself->VI_bounds.R_top = pself->TTP_vertPos;
    pself->VI_bounds.R_bottom = (pself->TTP_vertPos) + PIECE_HEIGHT;

} else {

    /* If the bounds are not on the game board, then reset the
     * current positions to be the original bounds. */

    pself->TTP_horizPos = pself->VI_bounds.R_left;
    pself->TTP_vertPos = pself->VI_bounds.R_top;
}

    /* Now, the game piece must draw itself at its newly-set
     * bounds (will draw itself over its original picture if the
     * new bounds were invalid). */

@call self::MSG_VIS_DRAW(0, gstate);

    /* Finally, destroy the temporary gstate used for drawing. */

GrDestroyState(gstate);
```

```
            /* Clear flag to indicate we are no longer dragging. */

        pself->TTP_dragging = FALSE;
    }
        /* Release the mouse grab now that the move has finished.
         * Other objects in the view (other game pieces, for example)
         * may now receive pointer, select, and drag events. */

    @call self::MSG_VIS_RELEASE_MOUSE();
}
```

23.4

Passive Grabs

MSG_VIS_ADD_BUTTON_PRE_PASSIVE,
MSG_VIS_REMOVE_BUTTON_PRE_PASSIVE,
MSG_VIS_ADD_BUTTON_POST_PASSIVE,
MSG_VIS_REMOVE_BUTTON_POST_PASSIVE

They type of mouse grab described in the previous section is also known as an “active” mouse grab. An object can also have two other types of mouse grabs: *pre-passive* and *post-passive*. While only one object may have the active mouse grab at any given time, any number of objects may have pre- or post-passive grabs on the mouse.

An object with a pre-passive mouse grab receives a copy of the input event before the event goes to its true destination. The copy will be a message of the form MSG_META_PRE_PASSIVE_... (e.g., if the event is MSG_META_PTR, the pre-passive event is MSG_META_PRE_PASSIVE_PTR). All pre-passive events will be handled before the true event is delivered to the object with the active grab. If any of the pre-passive copies of the event is returned with the flag MRF_PREVENT_PASS_THROUGH, the active event will not be sent to its destination. You can use this feature to set up input filters, having an object receive all mouse events pre-passive and filtering out those that should not be delivered. The message MSG_VIS_REMOVE_BUTTON_PRE_PASSIVE removes any pre-passive mouse grab from the object.

A post-passive mouse grab is similar to a pre-passive except that the post-passive object receives the event *after* it is sent to its normal destination. The post-passive object will receive a MSG_META_POST_PASSIVE_... with the same flags and position information as sent with the input event. An object that has the post-passive grab can release it by sending itself a MSG_VIS_REMOVE_BUTTON_POST_PASSIVE.

■ MSG_VIS_ADD_BUTTON_PRE_PASSIVE

```
void        MSG_VIS_ADD_BUTTON_PRE_PASSIVE();
```

A visible object sends itself this message (with **@call**) when it wants to gain a pre-passive mouse grab. The object will receive all subsequent mouse events in the form of MSG_META_PRE_PASSIVE_... *before* the event's destination object receives them; this message will carry the same input information as passed with the input event. The object should handle MSG_META_PRE_PASSIVE_... and, if the event is not to be sent normally to its destination, should return the flag MRF_PREVENT_PASS_THROUGH from that handler. The pre-passive grab will be released when the object sends itself a MSG_VIS_REMOVE_BUTTON_PRE_PASSIVE.

Source: Unrestricted.

Destination: Any visible object—typically sent by a visible object to itself when it wants to gain a pre-passive mouse grab.

Interception: Unlikely—Note that the object must, however, intercept any MSG_META_PRE_PASSIVE... events it is interested in.

■ MSG_VIS_REMOVE_BUTTON_PRE_PASSIVE

```
void        MSG_VIS_REMOVE_BUTTON_PRE_PASSIVE();
```

This message removes the pre-passive grab from a visible object.

Source: Unrestricted.

Destination: Any visible object—typically sent by a visible object to itself when it has a pre-passive grab and wants to release it.

Interception: Unlikely.

■ MSG_VIS_ADD_BUTTON_POST_PASSIVE

```
void        MSG_VIS_ADD_BUTTON_POST_PASSIVE();
```

A visible object sends itself this message (with **@call**) when it wants to gain the post-passive mouse grab. The object will receive all subsequent mouse events in the form of MSG_META_POST_PASSIVE_... *after* the event's destination object receives them; this message will carry the same input information as passed with the input event. The object should handle the appropriate MSG_META_POST_PASSIVE_... to properly receive the events.

The post-passive grab will be released when the object receives a MSG_VIS_REMOVE_BUTTON_POST_PASSIVE.

Source: Unrestricted.

Destination: Any visible object—typically sent by a visible object to itself when it wants to receive post-passive input events.

Interception: Unlikely—Note that the object must, however, handle any MSG_META_POST_PASSIVE_... events it is interested in.

23.4

■ MSG_VIS_REMOVE_BUTTON_POST_PASSIVE

```
void    MSG_VIS_REMOVE_BUTTON_POST_PASSIVE();
```

This message removes any post-passive mouse grab from the object.

Source: Unrestricted.

Destination: Any visible object—typically sent by a visible object to itself to release a post-passive mouse grab.

Interception: Unlikely.

The Gadget Exclusive and Focus Hierarchy

MSG_VIS_TAKE_GADGET_EXCL, MSG_VIS_RELEASE_GADGET_EXCL, MSG_VIS_LOST_GADGET_EXCL, MSG_VIS_VUP_QUERY_FOCUS_EXCL, MSG_VIS_FUP_QUERY_FOCUS_EXCL, MSG_VIS_VUP_ALLOW_GLOBAL_TRANSFER

The *gadget exclusive* is a marker in the system used by the UI to indicate which object has exclusive access to input gadget functionality. Only one gadget at a time may have the gadget exclusive, and the gadget exclusive may be forcibly taken from an object by another object (for example, when a system-modal dialog comes up during a drag operation).

At times, an object will force the mouse input to go to a different object from that which has the current active grab. This is done almost exclusively by the Specific UI library and is rarely, if ever, done by applications. Three messages can be used in determining which object has the current “gadget exclusive,” or active mouse grab. These are shown below.

In addition, individual objects can get information about the current state of the Focus hierarchy with MSG_VIS_VUP_QUERY_FOCUS_EXCL and MSG_VIS_FUP_QUERY_FOCUS_EXCL, also shown below.

To deal with a quick-transfer operation, a visible object may also have to use the message MSG_VIS_VUP_ALLOW_GLOBAL_TRANSFER. This message may be used to remove the mouse grab from the GenView in which the visible object resides, thereby allowing mouse events to reach other objects outside the view window.

■ MSG_VIS_TAKE_GADGET_EXCL

```
void      MSG_VIS_TAKE_GADGET_EXCL(
          optr   child);
```

This message causes the passed object to be given the gadget exclusive. Any object currently having the gadget exclusive will subsequently receive a MSG_VIS_LOST_GADGET_EXCL. This is used primarily by the Specific UI.

Source: Unrestricted. Typically sent by a visible object that will next grab the mouse.

Destination: Any visible object—normally sent by an object to its visible parent.

Parameters: *child* The optr of the object to which the gadget exclusive will be given (normally the calling object).

Return: Nothing.

Interception: Unlikely.

See Also: MSG_VIS_GRAB_MOUSE, MSG_VIS_LOST_GADGET_EXCL

■ MSG_VIS_RELEASE_GADGET_EXCL

```
void      MSG_VIS_RELEASE_GADGET_EXCL(
          optr   child);
```

This message causes the passed object to release the gadget exclusive. The object specified by *child* will then receive a MSG_VIS_LOST_GADGET_EXCL.

Source: Unrestricted. Typically sent by an object that has a mouse grab and expects to release it.

Destination: Any visible object—normally sent by the object to it's parent.

Parameters: *child* The optr of the object which is to lose the gadget exclusive (normally the calling object).

Return: Nothing.

Interception: Unlikely.

See Also: MSG_VIS_RELEASE_MOUSE, MSG_VIS_LOST_GADGET_EXCL

■ MSG_VIS_LOST_GADGET_EXCL

```
void    MSG_VIS_LOST_GADGET_EXCL();
```

This message, when received, indicates that the recipient has lost its hold on the gadget exclusive. When an object receives this, it should release the active mouse grab, if appropriate.

Source: Unrestricted—typically sent by the UI.

Destination: The visible object losing the gadget exclusive.

Interception: Mouse grabs should be released in this handler.

■ MSG_VIS_VUP_QUERY_FOCUS_EXCL

```
void    MSG_VIS_VUP_QUERY_FOCUS_EXCL(
        ObjectAndGrabParams *retValue);
```

This message queries up the visible hierarchy to find the object having the current focus in the current window. The current window is taken to be the window in which the object receiving the message resides.

Source: Unrestricted.

Destination: Any visible object—typically sent by a visible object to itself to find the focus optr in its window.

Parameters: *retValue* A pointer to an empty **ObjectAndGrabParams** structure, shown below.

Return: The **ObjectAndGrabParams** structure pointed to by *retVal* will be filled with the appropriate data.

Interception: Unlikely.

Structures: The structure of the **ObjectAndGrabParams** is shown below:

```
typedef struct {
    word    OAGP_grabFlags;  /* flags */
    word    OAGP_unused;     /* reserved */
    optr    OAGP_object;     /* object with focus */
} ObjectAndGrabParams;
```

The *OAGP_grabFlags* field contains a record of **HierarchicalGrabFlags**, defined in “Input,” Chapter 11 of the Concepts Book. The allowable flags are

HGF_SYS_EXCL Indicates this object has the system exclusive.

HGF_APP_EXCL Indicates this object has the application’s exclusive.

HGF_GRAB — Indicates this object wishes to grab the exclusive (if clear, the object wishes to release), when passed as a parameter. In the structure, it merely indicates that the object in *OAGP_object* has the exclusive.

HGF_OTHER_INFO — Twelve bits used by the specific UI depending on the type of hierarchical grab.

■ MSG_VIS_FUP_QUERY_FOCUS_EXCL

```
void    MSG_VIS_FUP_QUERY_FOCUS_EXCL(
        ObjectAndGrabParms *retValue);
```

This message queries the focus hierarchy to see which object has the current focus. The current focus object does not have to be in the caller's visible tree.

Source: Unrestricted.

Destination: Any visible object.

Parameters: *retValue* — A pointer to an empty **ObjectAndGrabParams** structure, shown above in the entry for MSG_VIS_VUP_QUERY_FOCUS_EXCL.

Return: The **ObjectAndGrabParams** structure pointed to by *retVal* will be filled with the appropriate data.

Interception: Unlikely.

Structures: The structure of the **ObjectAndGrabParams** is shown in the entry for MSG_VIS_VUP_QUERY_FOCUS_EXCL.

■ MSG_VIS_VUP_ALLOW_GLOBAL_TRANSFER

```
void    MSG_VIS_VUP_ALLOW_GLOBAL_TRANSFER();
```

This message is sent by a visible object to itself when a quick-transfer operation is underway and the pointer has to be allowed to leave the bounds of the object's view window. This message will travel up to the content object, which will indicate to the GenView that the pointer events must be allowed to go to other windows in the system.

Source: Unrestricted.

Destination: Any visible object—typically sent by a visible object to itself when the global transfer must be allowed outside its window.

Interception: Unlikely.

The Mouse Status

```
MSG_VIS_VUP_SET_MOUSE_INTERACTION_BOUNDS,
MSG_VIS_VUP_GET_MOUSE_STATUS,
MSG_VIS_VUP_TERMINATE_ACTIVE_MOUSE_FUNCTION,
MSG_VIS_VUP_BUMP_MOUSE
```

23.4

The UI can control the mouse image and events as they occur. Because the UI is based on the visible classes, the messages that control the mouse status are also available to you, though they will probably be of very little utility. These messages are given below.

■ MSG_VIS_VUP_SET_MOUSE_INTERACTION_BOUNDS

```
void    MSG_VIS_VUP_SET_MOUSE_INTERACTION_BOUNDS(@stack
        int   bottom,
        int   right,
        int   top,
        int   left);
```

A visible object requesting the mouse grab sends this message up the visible tree. This message will be handled by a content object. It indicates a new set of draggable bounds in case the user initiates a drag-scrolling operation. This message is sent automatically when the visible object requests the grab.

Source: A visible object requesting a mouse grab.

Destination: The sender sends this message to itself.

Parameters: *bottom, right, top, left*
The four bounds of the new bounding rectangle. Default is the bounds of the object sending the message.

Return: Nothing.

Interception: Unlikely.

■ MSG_VIS_VUP_GET_MOUSE_STATUS

```
word    MSG_VIS_VUP_GET_MOUSE_STATUS();
```

This message is rarely used by any objects outside the UI library. It returns the information passed with the last mouse event. The word return value contains two byte-sized sets of flags. The high byte is a list of the active UI functions, and the low byte represents the current button information.

Source: Unrestricted.

Destination: Any visible object—typically sent by a visible object to itself when it can not locally store the latest mouse information.

Parameters: None.

Return: A word containing two values: The high word is a record of type **UIFunctionsActive**, and the low word is a word of **ButtonInfo**.

Interception: Unlikely.

■ MSG_VIS_VUP_TERMINATE_ACTIVE_MOUSE_FUNCTION

```
void      MSG_VIS_VUP_TERMINATE_ACTIVE_MOUSE_FUNCTION();
```

This message is sent by a visible object to itself to terminate any active mouse function, forcing it to be a function of type “other.” This message is used only by the Specific UI library in cases where input synchronization problems occur in specific places. Applications should generally not use this message.

Source: A visible object handling active input events.

Destination: Sent by the visible object to itself.

Interception: This message should *not* be subclassed.

■ MSG_VIS_VUP_BUMP_MOUSE

```
void      MSG_VIS_VUP_BUMP_MOUSE(
          int    xBump,
          int    yBump);
```

This message causes the UI to bump the pointer image on the screen by the passed offsets as if the user had moved the mouse. It’s unlikely your objects will subclass this message; you may, however, use it to bump the mouse. It is most often used by specific UI objects (menus, scrollers) that make the mouse follow their movements.

Source: Unrestricted.

Destination: Any visible object.

Parameters:

xBump	The horizontal amount to bump the mouse.
yBump	The vertical amount to bump the mouse.

Return: Nothing.

Interception: Unlikely.

23.4.4.2 Keyboard Events

MSG_META_GRAB_KBD, MSG_META_FORCE_GRAB_KBD, MSG_META_RELEASE_KBD

The keyboard, like the mouse, may be grabbed by visible objects. Keyboard events arrive in the form MSG_META_KBD_CHAR; a single message represents all different types of keyboard events (unlike mouse events, which can be much more diverse).

To grab the keyboard, the visible object should send itself the message MSG_META_GRAB_KBD. If the grab should be made in any circumstances, the object should use MSG_META_FORCE_GRAB_KBD. To release the keyboard grab, the object should send itself MSG_META_RELEASE_KBD.

■ MSG_META_GRAB_KBD

```
void    MSG_META_GRAB_KBD();
```

A visible object will send itself this message (using **@call**) when it wants to gain the keyboard grab. To release the grab, it must send itself a MSG_META_RELEASE_KBD. If it wants to gain the exclusive regardless of whether another object has it, it should use MSG_META_FORCE_GRAB_KBD.

Source: Unrestricted.

Destination: Any visible object—typically sent by a visible object to itself to gain the keyboard grab.

Interception: Unlikely—Note that the object must, however, intercept MSG_META_KBD_CHAR to receive the subsequent keyboard events.

■ MSG_META_FORCE_GRAB_KBD

```
void    MSG_META_FORCE_GRAB_KBD();
```

A visible object will send itself this message (using **@call**) when it wants to gain the keyboard grab whether or not another object currently has it. To release the grab, the object must later send itself MSG_META_RELEASE_KBD.

Source: Unrestricted.

Destination: Any visible object—typically sent by a visible object to itself to force a keyboard grab.

Interception: Unlikely.

■ **MSG_META_RELEASE_KBD**

```
void      MSG_META_RELEASE_KBD();
```

A visible object that has the keyboard grab must send itself this message to release the grab.

Source: Unrestricted.

Destination: Any visible object—typically sent by a visible object to itself to release its keyboard grab.

Interception: Unlikely.

23.4.4.3 Ink and Pen Events

```
MSG_VIS_QUERY_IF_OBJECT_HANDLES_INK
```

If a visible object expects and wants Ink input, it should subclass the message MSG_VIS_QUERY_IF_OBJECT_HANDLES_INK. This message queries the visible object, which should return specific values based on the current input state and object bounds.

This message has two default handlers; the default **VisClass** handler always returns *false*, indicating that the visible object does not expect Ink input. You can, however, substitute a second default handler for the **VisClass** handler by adding the following line to your Goc file after the object's class definition:

```
@method VisObjectHandlesInkReply, <YourClass>,
                    MSG_VIS_QUERY_IF_OBJECT_HANDLES_INK;
```

This will, in essence, substitute the system-provided routine **VisObjectHandlesInkReply()** for a subclassed message handler. This routine will do the right thing for the object in interacting with the input manager. All subsequent input events will be considered Ink events.

■ **MSG_VIS_QUERY_IF_OBJECT_HANDLES_INK**

```
void      MSG_VIS_QUERY_IF_OBJECT_HANDLES_INK(
          VisCallChildrenInBoundsFrame  *data);
```

This message is subclassed by objects that handle Ink input. See above for directions on handling this message. The parameters are five data words that may be modified and the bounds of a rectangle. This message will be sent to all objects whose bounds fall within the passed rectangle.

Source: Unrestricted.

Destination: Any visible object.

Parameters: *data* A pointer to a **VisCallChildrenInBoundsFrame** structure containing the bounding rectangle and five words of data. This structure is described below.

Return: Three fields of the **VisCallChildrenInBoundsFrame** structure pointed to by *data* will have meaning on return. See the Interception paragraph below to determine whether the return values should be set or not in a subclassed handler.

VCCIBF_data1 Memory handle of the top object desiring ink.

VCCIBF_data2 Chunk handle of the top object desiring ink.

VCCIBF_data3 Upper bound of the object desiring ink.

Interception: If an object wants to receive ink messages, it should do the following things in its handler for this message: First compare the passed *VCCIBF_data3* value with the top bound of the object. Then, if the upper bound of the object is less than the *VCCIBF_data3* value (or if the value in *VCCIBF_data1* is zero), then the object should return the values indicated in the Return paragraph above. Otherwise, if these conditions are not met, it should return with the structures and registers unchanged.

Structures: The **VisCallChildrenInBoundsFrame** structure is shown below:

```
typedef struct {
      word        VCCIBF_data1;
      word        VCCIBF_data2;
      word        VCCIBF_data3;
      word        VCCIBF_data4;
      word        VCCIBF_data5;
            /* Five data words as described above */
      Rectangle   VCCIBF_bounds;
            /* bounds must be in the coordinate
             * system of the parent of the caller */
} VisCallChildrenInBoundsFrame;
```

Objects

23.5 Working with Visible Object Trees

Visible objects can not be used unless they are placed within object trees. The top object of the tree must be of a subclass of **VisContentClass**. Other objects in the tree can be of any visible object class. Keep in mind, however, that **VisClass** objects can not have children; only the composite classes (**VisCompClass** and **VisContentClass**) may have children. Therefore, **VisClass** is useful only for leaf objects.

Each visible object has a *VI_link* instance field. This field specifies the optr of that object's next sibling in the tree. Since this is a field of **VisClass**, both composites and noncomposites have it. Composite objects also have a *VCI_comp* field which contains the optr of the composite's first child. Thus, visible trees use the same format as normal object trees.

23.5.1 Creating and Destroying

MSG_VIS_DESTROY

Visible objects are just like any other objects in the ways they are created and destroyed. You can create visible objects in many ways; the easiest is to set them up in a resource and simply load or duplicate the resource block at run-time. You can also use the **ObjInstantiate()** routine to instantiate new visible objects on the fly from your predefined subclass of **VisClass**, **VisCompClass**, or **VisContentClass**. "GEOS Programming," Chapter 5 of the Concepts Book, details how to instantiate a new object of a given class.

Destroying visible objects is very simple. You only need to use the message MSG_VIS_DESTROY, which works on a visible object and all its children (if it has any). This message removes the visible branch from the screen if necessary, detaches the visible objects from the visible tree, and then destroys the object (freeing its instance chunk). The only thing the object must do is clean up after itself—in its MSG_META_DETACH or MSG_META_FINAL_OBJ_FREE handler, it should free any memory that it allocated explicitly for its own use, for example.

MSG_VIS_DESTROY also marks the destroyed object's parent invalid both in geometry and image so the tree will be updated properly.

■ MSG_VIS_DESTROY

```
void     MSG_VIS_DESTROY(
         VisUpdateMode updateMode);
```

This message is the high level routine for destroying branches of visible trees. This message will close and destroy the entire branch, unlinking any visible objects in the branch under the recipient object. The parent of the removed branch will be marked invalid for later visual update. This message may be subclassed to replace the geometry update handling, but this is not easily done and is rare.

Source: Unrestricted.

Destination: Any visible object (or head of a visible branch).

Parameters: *updateMode* A **VisUpdateMode** that determines when the visible parent of the destroyed object/branch will be updated on the screen.

Return: Nothing.

Interception: Unlikely—may be subclassed if the parent is not to be marked invalid. This is extremely rare.

23.5.2 Adding and Removing

MSG_VIS_ADD_CHILD, MSG_VIS_REMOVE, MSG_VIS_REMOVE_CHILD, MSG_VIS_MOVE_CHILD, MSG_VIS_ADD_NON_DISCARDABLE_VM_CHILD, MSG_VIS_REMOVE_NON_DISCARDABLE_VM_CHILD, MSG_VIS_REMOVE_NON_DISCARDABLE

Once you have objects instantiated, you can connect them together into an object tree. If you know the structure of your tree beforehand, you can create the tree explicitly in your **.goc** file, just as you would create your generic object tree.

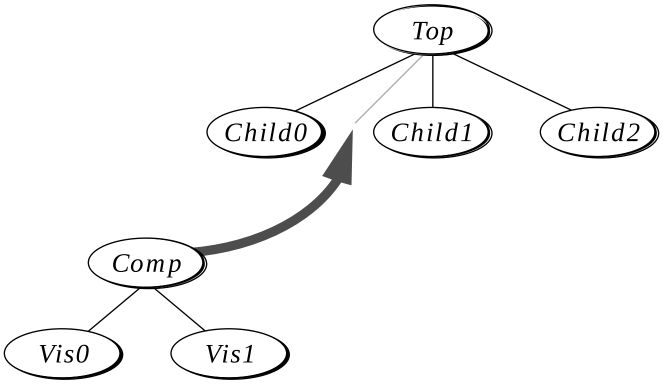
For many purposes, however, a dynamic object tree is much more useful. To use it, you must be able to add and remove objects easily as well as move them within the tree easily.

23.5.2.1 Adding and Removing Normally

To add an object to a branch of a visible tree, use the message MSG_VIS_ADD_CHILD. This message adds a visible object as the child of a composite; if the new child is a composite with its own children, the entire branch is added to the tree. You can add a child at any position in the composite's child list (e.g., as the first child, as the second child, as the last child, etc.), and you can mark the child dirty after addition if you want.

For example, if you have a composite visible object with two children that is currently unattached to any visible tree, you can attach it to a tree with MSG_VIS_ADD_CHILD. Figure 23-4 shows this process and the line of code used to add the composite as the second child of an object in the tree.

Note that the child in the example is added as child one. The child list of a composite is zero-based, so the first child is referred to as number zero, the second is number one, and so on. The last child may always be referred to as CCO_LAST; likewise, the first child may always be referred to as CCO_FIRST.



```
@call Top::MSG_VIS_ADD_CHILD(@Comp, 1|CCF_DIRTY);
```

Figure 23-4 *Adding a Visible Child*

Adding an entire visible branch into a visible tree is simple. The line of code above shows how to add the Comp object into the tree as the Top object's second child. (The labels act as the object names for this example.)

To remove an object or branch from a visible tree, you can use either MSG_VIS_REMOVE or MSG_VIS_REMOVE_CHILD. MSG_VIS_DESTROY may also be used, but only if the entire branch should be destroyed after being removed from the tree. Both MSG_VIS_REMOVE and MSG_VIS_REMOVE_CHILD detach the object from the visible tree and visually update the parent immediately. MSG_VIS_REMOVE, however, should be sent directly to the object being removed, while MSG_VIS_REMOVE_CHILD is sent to the parent of the object being removed. When removing a child from its parent, you can specify the child by position number (e.g., remove the last child or the first child).

You can move a child within its child list with MSG_VIS_MOVE_CHILD. This message is sent to the parent and simply moves the child within the parent's child list. To move a child from one parent to another, you must first remove it and then add it to the other parent with MSG_VIS_REMOVE_CHILD and MSG_VIS_ADD_CHILD. This message, as the others, will move the child's entire branch with it.

23.5.2.2 Adding and Removing Objects Not Saved to Documents

In many cases, you may save visible objects directly to VM documents using GenDocument objects and the document controllers. In cases like this, the objects you save to the file *must not* be discarded by the UI if the visible object is taken off the tree or removed. These objects are known as "non-discardable" visible objects stored in a VM file, and unless they're handled specially, they can cause your documents to crash your application.

Any top-level object you save directly to a file must be treated as non-discardable. That is, any object that gets dynamically added to or removed from a visible tree (such as a vis object that gets added as the child of a GenDocument when the document is opened) must be added, removed, and managed as a non-discardable object. There are three messages that deal specifically with non-discardable visible objects; these are detailed below and are MSG_VIS_ADD_NON_DISCARDABLE_VM_CHILD, MSG_VIS_REMOVE_NON_DISCARDABLE_VM_CHILD, and MSG_VIS_REMOVE_NON_DISCARDABLE.

■ MSG_VIS_ADD_CHILD

```
void        MSG_VIS_ADD_CHILD(
            optr                child,
            CompChildFlags      flags);
```

This message attaches the passed object as a child of the composite handling the message. If the parent is already opened and on the screen, you must invalidate the child with MSG_VIS_MARK_INVALID, passing the invalidation VOF_WINDOW_INVALID; the composite and child will be updated appropriately according to the **VisUpdateMode** passed with the invalidation. If, however, the parent is not opened, the child will automatically be opened when the parent is opened.

Source: Unrestricted.

Destination: Any visible composite object.

Parameters: *child* The optr of the visible object to be added as a child.

flags A structure of **CompChildFlags**, described below.

Return: Nothing.

Interception: You should not subclass this message.

Structures: The **CompChildFlags** structure contains one one-bit flag and one 15-bit unsigned numerical field. These are described below and can be found in **metaC.goh**.

```
typedef WordFlags CompChildFlags;
#define CCF_MARK_DIRTY          0x8000
#define CCF_REFERENCE           0x7fff

#define CCO_FIRST               0x0000
#define CCO_LAST                0x7FFF

#define CCF_REFERENCE_OFFSET         0
```

CCF_MARK_DIRTY If this flag is set, the object will be marked dirty when added to the tree.

CCF_REFERENCE The zero-based position of the new child. Other children will be shuffled forward in the child list as necessary. If greater than the number of children, it will be taken to be CCO_LAST.

Two special numbers may be used in the CCF_REFERENCE field:

CCO_FIRST Add as the first child.

CCO_LAST	Add as the last child.

Warnings: Do not pass the optr of an object that is already the child of another object. The results are unpredictable and will likely result in an error.

■ MSG_VIS_REMOVE

```
void        MSG_VIS_REMOVE(
            VisUpdateMode updateMode);
```

This is the high-level message that closes (if necessary) and removes a visible branch from the object tree. The parent of the branch is marked invalid for visual update according to the passed **VisUpdateMode**.

Unlike MSG_VIS_DESTROY, this message does not destroy the visible branch but only unlinks it from the tree. This message may not be subclassed by any object. This message is useful for hiding visible branches which may be added again later with MSG_VIS_ADD_CHILD.

Source: Unrestricted.

Destination: Any visible composite object.

Parameters:

updateMode	The **VisUpdateMode** used to update the composite from which the child is removed. VUM_MANUAL is not allowed.

Return: Nothing.

Interception: Do not subclass this message.

Warnings: This message will not allow VUM_MANUAL to be passed as the *updateMode.*

■ MSG_VIS_REMOVE_CHILD

```
void        MSG_VIS_REMOVE_CHILD(
            optr                child,
            CompChildFlags      flags);
```

This message removes the specified child from the object tree. This message should be rarely used and used with care; it does not close the visible branch but simply removes it. Consider using the higher-level MSG_VIS_REMOVE instead. Note that all grabs (focus, gadget, mouse, etc.) must be released by the child and all its children before the branch can be removed safely.

Source: Unrestricted.

Destination: The parent composite of the passed object.

Parameters: *child* The optr of the child to be removed. If this optr is not among the recipient's children, an error will likely occur.

flags The **CompChildFlags** as described in MSG_VIS_ADD_CHILD on page 1377.

Return: Nothing.

Interception: You should not subclass this message.

Warnings: Most likely you should use MSG_VIS_REMOVE instead of this message. MSG_VIS_REMOVE takes care of extra bookkeeping automatically that can be difficult.

■ MSG_VIS_MOVE_CHILD

```
void        MSG_VIS_MOVE_CHILD(
            optr                child,
            CompChildFlags      flags);
```

This message moves a child of the recipient to another location among its siblings. It essentially removes the child from the branch and then re-adds it in the same manner as MSG_VIS_ADD_CHILD. This message does *not* move the child from one parent to another; you must use a combination of MSG_VIS_REMOVE and MSG_VIS_ADD_CHILD to achieve that.

Source: Unrestricted.

Destination: The parent composite of the passed object.

Parameters: *child* The optr of the child to be removed. If this optr is not among the recipient's children, an error will likely occur.

flags The **CompChildFlags** as described in MSG_VIS_ADD_CHILD on page 1377.

Return: Nothing.

Interception: You should not subclass this message.

■ MSG_VIS_ADD_NON_DISCARDABLE_VM_CHILD

```
void        MSG_VIS_ADD_NON_DISCARDABLE_VM_CHILD(
            optr                child,
            CompChildFlags      flags);
```

This message performs exactly the same as MSG_VIS_ADD_CHILD except that it also increments the object's in-use count so it will never be discarded.

This is used on top objects in visible sub-trees that are saved to document files: The object is added as a non-discardable file when the document is opened, and it is removed as a non-discardable child (with MSG_VIS_REMOVE_NON_DISCARDABLE_VM_CHILD or MSG_VIS_REMOVE_NON_DISCARDABLE) when the document is closed.

Source: Unrestricted—typically a document object when opening the file.

Destination: The new parent of the visible object being added to the tree.

Parameters: *child* The optr of the new child being added to the tree.

flags A record of **CompChildFlags** indicating where the child should be added. CCF_MARK_DIRTY is ignored in this record.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_VIS_REMOVE_NON_DISCARDABLE_VM_CHILD

```
void      MSG_VIS_REMOVE_NON_DISCARDABLE_VM_CHILD(
          optr   child);
```

This message performs exactly the same as MSG_VIS_REMOVE_CHILD except that it is used with MSG_VIS_ADD_NON_DISCARDABLE_VM_CHILD rather than with MSG_VIS_ADD_CHILD. This message decrements the in-use count before removing the child, thereby undoing the extra increment performed by MSG_VIS_ADD_NON_DISCARDABLE_VM_CHILD.

Source: Unrestricted—typically a document object when closing the file.

Destination: The parent of the visible object being removed from the tree.

Parameters: *child* The optr of the child being removed.

Return: Nothing.

Interception: Generally not intercepted.

■ MSG_VIS_REMOVE_NON_DISCARDABLE

```
void      MSG_VIS_REMOVE_NON_DISCARDABLE(
          VisUpdateMode       updateMode)
```

This message performs exactly the same as MSG_VIS_REMOVE except that it decrements the object's in-use count before removing it. This message should therefore be used in conjunction with the adding message MSG_VIS_ADD_NON_DISCARDABLE_VM_CHILD and should not be used with

MSG_VIS_ADD_CHILD. Likewise, MSG_VIS_REMOVE should not be used with this MSG_VIS_ADD_NON_DISCARDABLE_VM_CHILD.

Source: Unrestricted—typically a document object when closing the file.

Destination: The visible object being removed from the tree.

Parameters: *updateMode* A **VisUpdateMode** value indicating when the object should be visibly removed from the tree. VUM_MANUAL is not allowed.

Return: Nothing.

Interception: Generally not intercepted.

23.5.3 Getting Visible Tree Information

MSG_VIS_FIND_CHILD, MSG_VIS_FIND_CHILD_AT_POSITION, MSG_VIS_COUNT_CHILDREN, MSG_VIS_FIND_PARENT

When an application manages a visible object tree, it will most likely need to find children, count children, or get the optr of a particular object in the tree. **VisClass** offers four messages to provide this bookkeeping information. These four messages are listed below.

■ MSG_VIS_FIND_CHILD

```
word      MSG_VIS_FIND_CHILD(
          optr   object);
```

This message returns the zero-based position of the specified child among the parent's children.

Source: Unrestricted.

Destination: Any visible composite object.

Parameters: *object* The optr of the child to be found.

Return: The zero-based position of the child in the parent's child list. (E.g., the first child returns zero, the second child returns one, the third child returns two, etc.) If *object* is not the optr of one of the composite's children, the value -1 will be returned. If sent to a non-composite object, the message will also return -1.

Interception: Unlikely.

■ MSG_VIS_FIND_CHILD_AT_POSITION

```
optr     MSG_VIS_FIND_CHILD_AT_POSITION(
         word   position);
```

This message returns the optr of the child occupying the *position* specified among the recipient's children.

Source: Unrestricted.

Destination: Any visible composite object.

Parameters: *position* The zero-based position of the child whose optr is to be returned.

Return: The optr of the child occupying the passed *position*. If no child occupies that position, or if the message is sent to a non-composite object, the retuned value will be NullOptr.

Interception: Unlikely.

■ MSG_VIS_COUNT_CHILDREN

```
word     MSG_VIS_COUNT_CHILDREN();
```

This message returns the number of children the recipient object has. The count is not zero-based; if a composite has five children, this message will return five.

Source: Unrestricted.

Destination: Any visible composite object.

Parameters: None.

Return: The number of children the object has. If the object has no children (includes non-composite objects), the return will be zero.

Interception: Unlikely.

■ MSG_VIS_FIND_PARENT

```
optr     MSG_VIS_FIND_PARENT();
```

This message returns the optr of the recipient's parent composite.

Source: Unrestricted.

Destination: Any visible composite object.

Parameters: None.

Return: The optr of the object's parent composite. If the message is sent to the top node of a visible tree, the return value will be NullOptr.

Interception: Unlikely.

Tips: If you want to send a message to your parent, use the **@visParent** macro defined in the next section.

23.5.4 Sending Messages Through the Tree

Often an object will need to contact its parent or children but will not know their precise optrs. Other times, you may need to send a message to a particular object in the tree but don't know where in the tree the object is. GEOS provides several ways to dispatch messages to parents and children and even to objects of a given class.

23.5.4.1 Contacting Parents and Children Directly

```
@visParent, @visChildren, MSG_VIS_CALL_PARENT,
MSG_VIS_SEND_TO_PARENT, MSG_VIS_SEND_TO_CHILDREN
```

To send a given message to an object's parent (to check valid bounds, to check the parent's state, etc.), you can use the macro **@visParent**. To do this, substitute the macro in place of the destination object's name, as follows:

```
      kids = @call @visParent::MSG_VIS_COUNT_CHILDREN();
```

The above call is more efficient than using two message calls for the same thing, as follows:

```
      myParent = @call self::MSG_VIS_FIND_PARENT();
      kids = @call myParent::MSG_VIS_COUNT_CHILDREN();
```

Note that either **@call** or **@send** may be used with the **@visParent** macro.

A similar macro, **@visChildren**, is offered for sending messages to a composite's children. Note, however, that you can only use the **@send** keyword to send messages with this macro; there is no way for a single call to collect return values from many different objects. So, if you need to pass pointers to your children, you will have to get each child's optr in turn and dispatch the message to it individually. (Recall that pointers should not be passed with **@send** because they may be invalidated by the time the message is handled.) The format for using the **@visChildren** macro is as follows:

```
@send @visChildren::MSG_VIS_INVALIDATE();
```

This usage is much more efficient than counting the composite's children and using a loop to get the child's optr and send the child the message.

In addition to providing the above macros, **VisClass** lets you pass recorded events to a visible object's parent or children. This can be useful, for example, if you need to send a message to another object's parent; you might otherwise first have to find the optr of the object's parent and then send the message directly. Instead, you can use the **VisClass** messages MSG_VIS_CALL_PARENT and MSG_VIS_SEND_TO_PARENT to pass recorded (encapsulated) events to the recipient's parent.

To send a recorded event to a visible object's children use the message MSG_VIS_SEND_TO_CHILDREN. The recorded event will not be allowed to return anything for the same reason the **@visChildren** macro can not.

■ MSG_VIS_CALL_PARENT

```
void    MSG_VIS_CALL_PARENT(
        EventHandle event);
```

This message delivers the passed recorded event to the parent of the recipient object. It acts as a call, executing immediately even across threads.

Source: Unrestricted.

Destination: Any visible object.

Parameters: *event* The event handle of the prerecorded event to be delivered.

Return: Nothing.

Interception: Unlikely.

Tips: In Goc, an object should not send this message to itself; instead, it should use the **@visParent** macro described above.

■ MSG_VIS_SEND_TO_PARENT

```
void      MSG_VIS_SEND_TO_PARENT(
          EventHandle event);
```

This message delivers the passed recorded event to the parent of the recipient object. It acts as a send, allowing the caller to continue execution without waiting for the event to be handled.

Source: Unrestricted.

Destination: Any visible object.

Parameters: *event* The event handle of the recorded event to be delivered.

Return: Nothing.

Interception: Unlikely.

Tips: In Goc, an object should not send this message to itself; instead, it should use the **@visParent** macro described above.

■ MSG_VIS_SEND_TO_CHILDREN

```
void      MSG_VIS_SEND_TO_CHILDREN(
          EventHandle event);
```

This message delivers the passed recorded event to each of the recipient's children. It acts as a message send and can not return values; there is no way for a single message to call several objects and return values from each with a single call.

Source: Unrestricted.

Destination: Any visible composite object.

Parameters: *event* The event handle of the recorded event to be delivered.

Return: Nothing.

Structures: In Goc, an object should not send this message to itself; instead, it should use the **@visChildren** macro described above.

Warnings: Do not pass pointers in the recorded event's parameters because the message may not be handled before the pointers are invalidated.

23.5

23.5.4.2 Sending Classed Events Up the Tree

```
MSG_VIS_VUP_FIND_OBJECT_OF_CLASS,
MSG_VIS_VUP_CALL_OBJECT_OF_CLASS,
MSG_VIS_VUP_SEND_TO_OBJECT_OF_CLASS,
MSG_VIS_VUP_TEST_FOR_OBJECT_OF_CLASS
```

VisClass has several messages that automatically travel up the visible object tree until they get to an object that meets the proper criteria; they then deliver themselves to that object. This is accomplished by the default handler in **VisClass** recognizing whether the given object meets the criteria; if it does not, the handler passes the message up to the object's parent.

To find an object of a given class in the tree, use the message MSG_VIS_VUP_FIND_OBJECT_OF_CLASS. This message travels up the tree starting at the recipient until it reaches the first object of that class. So if you are looking for an object of **VisCompClass**, at the most the message will be passed up a single level.

If you need to send a message to an object of a given class in the visible tree, you can either find it and send it the message or send the message directly with the visible upward messages MSG_VIS_VUP_CALL_OBJECT_OF_CLASS and MSG_VIS_VUP_SEND_TO_OBJECT_OF_CLASS. These messages search up the tree in the same was as MSG_VIS_VUP_FIND_OBJECT_OF_CLASS, and they then deliver the passed event to the resulting object.

To see simply whether an object of a given class exists in the visible tree, use MSG_VIS_VUP_TEST_FOR_OBJECT_OF_CLASS. This is useful if you are going to use MSG_VIS_VUP_CALL_OBJECT_OF_CLASS (or its counterpart). If no object of the given class exists, you can skip sending the recorded message.

To send classed messages to an object in the tree that is of a specific class, you can use **TravelOption** with certain **MetaClass** messages. **VisClass** defines a travel option TO_VIS_PARENT, which sends the message to the first object up the tree that is of the specified class.

■ MSG_VIS_VUP_FIND_OBJECT_OF_CLASS

```
optr      MSG_VIS_VUP_FIND_OBJECT_OF_CLASS(
          ClassStruct *class);
```

This message searches up the visible object tree until it encounters an object of the specified class. It then returns the optr of that object.

Source: Unrestricted.

Destination: Any visible object—typically sent by a visible object to itself to find an object of the given class above it in the visible object tree.

Parameters: *class* A pointer to the **ClassStruct** structure of the class to be searched for.

Return: The optr of the object of the passed class. If multiple objects of this class exist in the tree, the first encountered will be returned. If no objects of this class are in the tree, a NullOptr will be returned.

Interception: Unlikely—only objects that masquerade as objects of a different class would subclass this message. This practice is highly discouraged.

■ MSG_VIS_VUP_TEST_FOR_OBJECT_OF_CLASS

```
Boolean  MSG_VIS_VUP_TEST_FOR_OBJECT_OF_CLASS(
         ClassStruct *class);
```

This message searches up the visible tree and determines whether an object of the given class is in the tree.

Source: Unrestricted.

Destination: Any visible object—typically sent before MSG_VIS_VUP_CALL_OBJECT_OF_CLASS or MSG_VIS_VUP_SEND_TO_OBJECT_OF_CLASS to ensure that a recipient for the message exists.

Parameters: *class* A pointer to the **ClassStruct** structure of the class to be searched for.

Return: *True* if the class is found; *false* if it is not.

Interception: Unlikely.

■ MSG_VIS_VUP_CALL_OBJECT_OF_CLASS

```
void     MSG_VIS_VUP_CALL_OBJECT_OF_CLASS(
         EventHandle event);
```

This message searches up the visible tree until it encounters an object of the proper class for the recorded event. The class is specified within the passed *event*. When the first such object is found, it will be called with the classed event as if an **@call** had been used with the event directly.

Source: Unrestricted.

Destination: Any visible object—The passed event will be delivered to the first object of the appropriate class, not necessarily to the recipient of the MSG_VIS_VUP_CALL_OBJECT_OF_CLASS.

Parameters: *event* The event handle of a recorded event.

Return: The classed event delivered by this message may return values as determined by the event. This message returns nothing.

Interception: Unlikely.

■ MSG_VIS_VUP_SEND_TO_OBJECT_OF_CLASS

```
void        MSG_VIS_VUP_SEND_TO_OBJECT_OF_CLASS(
            EventHandle event);
```

This message searches up the visible tree until it encounters an object of the proper class for the recorded event. The class is specified within the passed *event*. When the first such object is found, the message will be delivered to that object as if an **@send** had been used with the event directly.

Source: Unrestricted.

Destination: Any visible object—The passed event will be delivered to the first object of the appropriate class, not necessarily to the object that received the MSG_VIS_VUP_SEND_TO_OBJECT_OF_CLASS.

Parameters: *event* The event handle of a recorded event.

Return: Nothing. The recorded event may not return anything.

Interception: Unlikely.

Warnings: The recorded event should not pass pointers among its parameters.

23.5.4.3 Visible Upward Queries

MSG_VIS_VUP_QUERY

You can create your own messages that get passed up the visible tree by creating aliases of MSG_VIS_VUP_QUERY. MSG_VIS_VUP_QUERY, by itself, does nothing useful; it simply gets passed up the visible tree without ever being handled.

You can create your own visible queries to return different types of values from objects higher up in the object tree. Take, for example, a music-teaching application which uses note objects as distant children of a sheet music object. If the note objects needed to know what key they were supposed to be in, they would query up the tree with a special upward query message. If we assume the sheet music object kept this information, it would want to respond to the query message with the appropriate value. Thus, the sheet music class would have a line similar to the following:

```
@alias(MSG_VIS_VUP_QUERY_MUSIC_KEY)
        MusicKeyType MSG_VIS_VUP_QUERY();
```

This line creates an alias for the general-purpose upward query. The sheet music object should have a handler for MSG_VIS_VUP_QUERY_MUSIC_KEY in which it figures out and returns the appropriate key. This type of upward query can be invaluable in many different situations.

■ MSG_VIS_VUP_QUERY

```
void        MSG_VIS_VUP_QUERY();
```

This message simply queries up the tree until it is handled. It is extensible so visible objects can implement their own upward queries without adding handlers to every class in between it and the query handler.

It is extremely rare that any object would send or handle MSG_VIS_VUP_QUERY on its own; many classes, however, may alias this message and create their own versions of the upward query.

Source: Unrestricted—see note above.

Destination: Any visible object—see note above. Typically sent by a visible object either to itself or directly to its visible parent.

Parameters: None—an alias of this message may have parameters.

Return: Nothing—an alias of this message may have return values.

Interception: MSG_VIS_VUP_QUERY itself should not be intercepted. Its aliases, however, should be intercepted by the appropriate classes that will handle them.

23.5.4.4 Sending Messages to Window Groups

MSG_VIS_VUP_CALL_WIN_GROUP, MSG_VIS_VUP_SEND_TO_WIN_GROUP

VisClass offers two specific messages for contacting window group objects in a visible tree. These are similar to MSG_VIS_VUP_CALL_OBJECT_OF_CLASS and MSG_VIS_VUP_SEND_TO_OBJECT_OF_CLASS, except they find the window group of the recipient and deliver the event to the window object. These two messages are detailed below.

■ MSG_VIS_VUP_CALL_WIN_GROUP

```
void      MSG_VIS_VUP_CALL_WIN_GROUP(
          EventHandle event);
```

This message searches up the visible object tree until it encounters a window group object. When the first window group is found, it will be called with the classed event as if an **@call** had been used.

Source: Unrestricted—typically sent by a visible object to itself to deliver the event to its window.

Destination: Any visible object—the passed event will be delivered to a window object, not necessarily to the object that receives the MSG_VIS_VUP_CALL_WIN_GROUP.

Parameters: *event* The event handle of a recorded event to be delivered to the window object.

Return: The passed event may return its own values, but MSG_VIS_VUP_CALL_WIN_GROUP returns nothing.

Interception: Unlikely.

See Also: MSG_VIS_VUP_CALL_OBJECT_OF_CLASS.

■ MSG_VIS_VIS_SEND_TO_WIN_GROUP

```
void      MSG_VIS_VIS_SEND_TO_WIN_GROUP(
          EventHandle event);
```

This message searches up the visible object tree until it encounters a window group object. When the first window group is found, the passed event will be delivered to that object as if an **@send** had been used directly.

Source: Unrestricted—typically sent by a visible object to itself to deliver the event to its window.

Destination: Any visible object—the passed event will be delivered to a window object, not necessarily to the object that receives the MSG_VIS_VUP_CALL_WIN_GROUP.

Parameters: *event* The event handle of a recorded event to be delivered to the window object.

◆ Objects

Return: Nothing. The recorded event may not return values.

Interception: Unlikely.

Warnings: The recorded event may not pass pointers among its parameters.

See Also: MSG_VIS_VUP_SEND_TO_OBJECT_OF_CLASS.

23.5.5 Visible Object Window Operations

MSG_VIS_OPEN_WIN, MSG_VIS_CLOSE_WIN,
MSG_VIS_WIN_ABOUT_TO_BE_CLOSED, MSG_VIS_MOVE_RESIZE_WIN

Typically, windows will be managed entirely by generic UI objects and the Specific UI library currently in use. You can, however, manage your own **VisClass**-based window objects by using messages normally used only by Specific UI objects. This is a difficult and complex task, however, and it is not recommended. The four messages used for this purpose are detailed below.

■ MSG_VIS_OPEN_WIN

```
void    MSG_VIS_OPEN_WIN(
        WindowHandle parentWindow);
```

This message is sent to a window or portal object to open its window. The graphics window will be opened and drawn on the screen, and a visual update will propagate down the displayed visible tree.

Source: Visual update mechanism.

Destination: A window group (VTF_IS_WINDOW) or portal (VTF_IS_PORTAL) object.

Parameters: *parentWindow* The window handle of the open window in which the new window will be created.

Return: Nothing.

Interception: The window group or portal must intercept in order to open the new graphics window and set initial parameters (*VCI_window*).

■ MSG_VIS_CLOSE_WIN

```
void    MSG_VIS_CLOSE_WIN();
```

This message is sent to a window or portal object that has its graphics window currently on the screen. The window and its visible branch will be closed (taken off the screen).

Source: Visual update mechanism.

Destination: A window group (VTF_IS_WINDOW) or portal (VTF_IS_PORTAL) object.

Interception: The window group or portal may intercept if more needs to be done than simply closing the graphics window.

■ MSG_VIS_WIN_ABOUT_TO_BE_CLOSED

```
void        MSG_VIS_WIN_ABOUT_TO_BE_CLOSED();
```

23.5

This message notifies the recipient that the window it's displayed in is about to be closed. The default handler for this will remove the visible branch of the window off the screen so no redraws will occur before the window closes.

Source: Visual update mechanism.

Destination: The window group (VTF_IS_WINDOW) or portal (VTF_IS_PORTAL) object that is being closed; the message will then propagate down the visible tree.

Interception: May be intercepted by either the window group or the VisContent object if the default behavior is not appropriate for the object.

■ MSG_VIS_MOVE_RESIZE_WIN

```
void        MSG_VIS_MOVE_RESIZE_WIN();
```

This message causes the window object to move and/or resize itself based on the bounds of its content object. This message is used only by window or portal objects and is rarely, if ever, used by objects in your applications.

Source: Part of the visual update mechanism—typically called by the window object on itself in its MSG_VIS_UPDATE_WINDOWS_AND_IMAGE handler.

Destination: The window group (VTF_IS_WINDOW) or portal (VTF_IS_PORTAL) object to be resized or moved.

Interception: The window object may intercept if it does not want to resize to the bounds of its content object.

23.6 Visible Layers and 32-Bit Graphics

```
MSG_VIS_LAYER_SET_DOC_BOUNDS,
MSG_VIS_LAYER_GET_DOC_BOUNDS,
MSG_VIS_LAYER_INSERT_OR_DELETE_SPACE
```

The GEOS graphic space is built on a basis of 16-bit coordinates. A few applications, however, will require much more space for their documents. GEOS also supports the special use of 32-bit coordinates; applications using these coordinates are said to be using the "large document" model.

The visible object library was originally designed with 16-bit coordinates in the default GEOS graphics space. All instance data, message parameters and return values, and coordinates are assumed to be word-sized integers. A large-document visible object library is not provided primarily because there is no single visible model that works well with large coordinates, and also because no single large model works efficiently for all applications.

It is possible, however, for applications to subclass the standard visible classes in order to get them to support the large document model. In order to do this, however, you must understand the basic issues involved with the large model and with using 32-bit coordinates.

As the application developer, you should read the following sections if you will be using the large document model. Each one deals with a different problem facing applications that want large documents.

In addition, some complex applications may offer several different visible layers in the program. For example, a spreadsheet might want to include a graphic object layer for graphics and charts. The content object of the visible tree can be set up to have several different children, each of which is a "layer" that manages its own objects in the window.

The content in charge of the layers must occasionally set and retrieve the document bounds of its children. For example, if the user wants to set the page size of the spreadsheet that has a graphical layer, the content object must set the new size for all its children. To retrieve the current document bounds of a particular object, send it a MSG_VIS_LAYER_GET_DOC_BOUNDS.

To set the document bounds, send a MSG_VIS_LAYER_SET_DOC_BOUNDS. In addition, the application may add or delete space into a layer with MSG_VIS_LAYER_INSERT_OR_DELETE_SPACE; for example, if a spreadsheet needs to resize a column, it must send this message to the graphic layer to ensure the layers handle the sizing properly.

■ MSG_VIS_LAYER_SET_DOC_BOUNDS

```
void      MSG_VIS_LAYER_SET_DOC_BOUNDS(@stack
          sdword bottom,
          sdword right,
          sdword top,
          sdword left);
```

This message sets the 32-bit document bounds for a particular layer object. This message is typically sent by a VisContent object to its children, which are assumed to be layer objects.

Source: Unrestricted—typically sent by a VisContent to its large children.

Destination: Any large (32-bit) visible object or composite.

Parameters: *all* The four parameters describe the new bounds of the large document. Since layer objects typically cover the entire document, the layer object will set its bounds to these values.

Return: Nothing.

Interception: All layer objects and large objects must subclass this message. There is no default handler.

■ MSG_VIS_LAYER_GET_DOC_BOUNDS

```
void      MSG_VIS_LAYER_GET_DOC_BOUNDS(
          RectDWord *bounds);
```

This message returns the 32-bit document bounds of a particular large visible object. Typically, a VisContent will query its large children with this message when it needs to know their bounds.

Source: Unrestricted—typically sent by a VisContent to its large children.

Destination: Any large (32-bit) visible object or composite.

Parameters: *bounds* A pointer to an empty **RectDWord** structure that describes a set of large bounds.

Return: The pointer to the filled **RectDWord** structure. The structure should, upon return, contain the 32-bit bounds of the large visible object.

Interception: All layer objects and large objects must subclass this message. There is no default handler.

Structures: The **RectDWord** structure is show below for convenience:

```
typedef struct {
    sdword   RD_left;
    sdword   RD_top;
    sdword   RD_right;
    sdword   RD_bottom;
} RectDWord;
```

■ MSG_VIS_LAYER_INSERT_OR_DELETE_SPACE

```
void     MSG_VIS_LAYER_INSERT_OR_DELETE_SPACE()
```

This message should be sent by one layer to another when it changes sizes. An example would be a spreadsheet application with a graphic layer; when a column in the spreadsheet resizes, the graphic layer must resize accordingly.

Source: Any visible layer object.

Destination: The visible layer object requiring resizing.

Parameters: None.

Return: Nothing.

Interception: This message must be intercepted; it has no default behavior.

23.6.1 Using Visible Document Layers

The first issue with large documents is the basic structure of the visible tree. The topmost object of a large visible tree must be able to keep track of information about the current translations and coordinates being used. Typically, the topmost object would be a subclass of **VisContentClass**; this does not change with large trees, though another consideration has to be taken care of.

A content object should not contain any information about the document itself; the content object should not be stored along with other document information. Thus, the object that acts as the content can not be the object that also manages the 32-bit document space. Instead, it is easiest to use a **VisCompClass** object to manage the document space and to place this object as a child of the content.

In addition, you can set up several different composite children of the content, each one managing its own large document space. Each one of these composites is called a "layer object" and manages a "layer" of the document. This way, you can have several different layers (such as a spreadsheet layer and a graphic layer) in the same document. The layer object should manage the calculations and translations necessary for supporting the large document space.

23.6.2 Using 16-Bit Drawing Commands

All the GEOS drawing commands are based on 16-bit coordinates. Drawing may only occur within a 16-bit graphics space. The graphics system does, however, support extended translations of the 16-bit graphics space within 32-bit coordinates.

This means that you can think of the large document space as a number of smaller, 16-bit "local" graphics spaces. When you need to draw anything, you

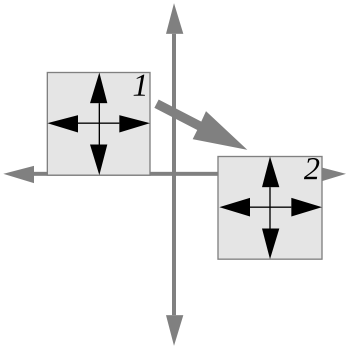


Figure 23-5 *Drawing in a Large Document*

Both of the shaded coordinate spaces represent a "local" 16-bit graphics space within the larger, 32-bit space. The application draws first in the shaded area labeled "1." It then does an extended translation, moving the origin of the local 16-bit space to "2." It can then execute all graphics commands as if the origin were at the new origin of the local space.

first use an extended translation to get near your drawing point in the large document. Then you can draw using 16-bit offsets from the point to which you translated. This is illustrated in Figure 23-5.

The application using the extended translation can do so in two different ways: It can store a 32-bit translation with each visible object, or it can store a 32-bit translation in a visible composite, thereby making the entire visible branch under the composite in a 32-bit graphics space. Visible trees implementing layer objects must use the second option; the layer must contain the translation for the entire visible branch that it manages. If a simple large visible tree does not need layers, it may wish to use the first solution.

The first of the two solutions is simple but takes up additional memory and time; each object, when it draws, must translate the coordinates, draw, then translate back. Two translations for each object could significantly affect drawing performance unless visual updates are handled carefully.

The other solution, storing a translation in the composite object, requires just two translations for the entire branch headed by the composite. Whenever the composite receives MSG_VIS_DRAW or MSG_VIS_VUP_CREATE_GSTATE, it executes the translation on the GState. As long as the translation is always relative to the GState's current transformation (done using **GrApplyTranslationDWord()**), and as long as the translation is undone before the GState is returned to the caller, drawing will be done properly.

23.6.3 The 16-Bit Limit on Visual Bounds

Individual visual objects have 16-bit bounds. If the rules of drawing in the 32-bit graphics space are followed, this is not a problem. Individual objects are constrained to 16-bit width and 16-bit height, however.

The only objects for which having 16-bit bounds is a problem are the tree's content object and all the layer objects. This is because these two types of objects must have width and height greater than 16 bits. This problem is handled differently for each object:

For the content object, the *VI_bounds* field is set to all zeros and is not used. Normally, the content uses its bounds to set the document size in the

GenView; the responsibility for setting the GenView's document size then falls to the application.

For the layer object, the *VI_bounds* field is set to all zeros and is not used. If the bounds of the layer object are important to the application, the layer object must be given new instance fields to hold the 32-bit coordinates of the bounds. If not, nothing needs to be done; the bounds can simply be ignored.

23.6

23.6.4 Handling MSG_VIS_DRAW

In normal-sized visible trees, MSG_VIS_DRAW gets sent to all objects in the tree. This draws the entire visible tree, no matter what portions are visible on the screen. For large documents, this can quickly get unwieldy and inefficient.

A better solution is for the layer object of each particular layer to intercept MSG_VIS_DRAW and pass it on only to those objects in the affected portion of the document. For this to work, of course, the layer object must know something of the structure and composition of the visible branch below it. This is the primary reason the visible object library does not offer a standard visible layer class—in essence, it must be defined specifically for each application.

23.6.5 Managing 32-Bit Geometry

Because visible objects and composites maintain their 16-bit bounds in a large document, they can still use the 16-bit geometry manager. Layer objects, however, can not use the geometry manager. All geometry management on the layer level (child positioning, resizing, etc.) must be done by the layer object itself in conjunction with its child composites.

23.6.6 Handling Mouse Events

The GenView is built to support large documents, though its default is normal-sized documents. Thus, you can request the view to send large mouse events, which carry with them 32 bits of integer and 16 bits of fraction in each dimension. The visible classes have no default handlers for these messages, so your own classes will have to handle them appropriately.

The large mouse events are the equivalent of their 16-bit counterparts. They just carry different data to represent the coordinates of the pointer. All the mouse events, both large and normal, are described in “Input,” Chapter 11 of the Concepts Book.

Normal composite objects will not be affected by large mouse events; the composite will continue to send the event on to the proper child as if it were in a 16-bit document. Layer and content objects, however, must process these messages differently.

If no object within the visible tree has the mouse grabbed, the content of the tree must decide which layer object should receive the mouse events. How the content and layer objects negotiate this is up to the particular application; some applications may consider that one layer is “on top of” the others, and other applications may determine which layer is the most appropriate based on the event context. By default, the large mouse events are passed by **VisContentClass** straight on to the first layer child. If you want a more complex scheme, you will have to subclass **VisContentClass** and intercept these messages.

23.6.7 Setting Up the Objects

Finally, you should be aware of what attributes to set in both the GenView and the VisContent objects of your visible tree to support the large document. You could easily find this out by looking through the various attributes of those classes (and you are encouraged to do so), but the following guide is provided for convenience:

In the GenView object, set GVA_WINDOW_COORDINATE_MOUSE_EVENTS in the *GVI_attrs* instance data field. This will pass mouse events in terms of offsets from the upper-left corner of the view window rather than in absolute document coordinates. **VisContentClass** handles these events and automatically turns them into document coordinates.

In the VisContent, set VCNA_WINDOW_COORDINATE_MOUSE_EVENTS and VCNA_LARGE_DOCUMENT_MODEL in the *VCNI_attrs* field. The first of these tells the content that it will receive input events in window coordinates rather than in document coordinates. The second will have the effect of making the content object ignore its bounds field and geometry management. It will also tell the content that all its children are layer objects. Setting the large document model will cause the content to send all the following messages on to all its children rather than handle them directly: MSG_VIS_DRAW, MSG_VIS_CONTENT_VIEW_ORIGIN_CHANGED, MSG_VIS_CONTENT_VIEW_SCALE_FACTOR_CHANGED, and MSG_VIS_CONTENT_VIEW_SIZE_CHANGED.

You do not have to create all the objects you may need; be aware that the Graphic Object Library provides its own layer object that supports a 32-bit graphic object layer. The spreadsheet object also exports its own layer object.

23.6.7.1 Requirements of Layer Objects

Layer objects will need to handle MSG_VIS_LAYER_SET_DOC_BOUNDS, which will be sent to all layers when the document bounds change.

In essence the layer object's job is to isolate its children and their children from the fact that they are in a 32-bit document rather than a 16-bit document. In order to do this, layer objects must do several things:

- Store the translation
  This is a set of 32-bit translation offsets. These offsets represent the current location, if any, where the layer is being drawn in the 32-bit coordinate space.
- Intercept MSG_VIS_DRAW
  In its handler, the layer should apply any 32-bit translations (using **GrApplyTranslationDWord()**) on the GState provided by MSG_VIS_DRAW. It should then intelligently select which of its children are affected by the drawing event and pass the message on to them with the translated GState. The handler should then untranslate the GState before returning.
- Intercept MSG_VIS_VUP_CREATE_GSTATE
  In the handler, the layer must *first* call its superclass (to create the GState normally), then apply the necessary translation using **GrApplyTranslationDWord()**. It can then return the translated GState.

- Intercept and handle large mouse events
  Each large mouse event that may be interesting to the layer must be handled by the layer object. This is to ensure that mouse events that occur when no object has the mouse grab get handled. If the layer has 16-bit visible children, it should translate the mouse event into small mouse events before passing it to its superclass. If the layer has 32-bit visible children, it should first translate both mouse coordinates and then decide which of its large children should receive the event.

23.6.7.2 Requirements of 32-bit Visible Objects

Large visible objects in the visible tree should be equipped to handle both large and small mouse events. This is because the parent object may have both 32-bit and 16-bit children. If it does, it will have to send out only small mouse events; otherwise, the 16-bit objects would be confused by the large mouse events.

23.6.7.3 Requirements of 16-bit Visible Objects

Because 16-bit objects are the norm and not the exception, there are few requirements for them. However, 16-bit objects in 32-bit documents must do one thing: they must prevent the system from allowing the user to select the object and then drag-scroll too far. To prevent this from happening, the visible object must call MSG_VIS_VUP_SET_MOUSE_INTERACTION_BOUNDS to set a temporary boundary on drag scrolling.

23.7 VisClass Error Checking

```
MSG_VIS_VUP_EC_ENSURE_WINDOW_NOT_REFERENCED,
MSG_VIS_VUP_EC_ENSURE_OBJ_BLOCK_NOT_REFERENCED,
MSG_VIS_VUP_EC_ENSURE_OD_NOT_REFERENCED
```

VisClass provides three messages you can use for special error checking. These three messages are provided especially to make sure that all objects being destroyed or removed will be destroyed cleanly. The default handlers for these messages should cause a fatal error if their criteria are not met.

■ MSG_VIS_VUP_EC_ENSURE_WINDOW_NOT_REFERENCED

```
void	MSG_VIS_VUP_EC_ENSURE_WINDOW_NOT_REFERENCED(
	WindowHandle window);
```

This message travels up the visible object tree, ensuring that the window handle passed is not stored anywhere that it might cause trouble. This message may be used when a window is being destroyed and its handle is about to be freed.

Source: Unrestricted.

Destination: Any visible object in the affected visible tree.

Parameters: *window* The handle of the window being destroyed.

Return: Nothing—if it returns, no error was encountered.

Interception: Do not intercept this message.

■ MSG_VIS_VUP_EC_ENSURE_OBJ_BLOCK_NOT_REFERENCED

```
void	MSG_VIS_VUP_EC_ENSURE_OBJ_BLOCK_NOT_REFERENCED(
	MemHandle objBlock);
```

This message travels up the visible object tree, ensuring that the memory handle (the handle of an object block) passed is not stored anywhere that it might cause trouble. This message may be used just before the object block's handle is freed to ensure that no stale references will remain.

Source: Unrestricted.

Destination: Any visible object in the affected visible tree.

Parameters: *objBlock* The handle of the block being freed.

Return: Nothing—if it returns, no error was encountered.

Interception: Do not intercept this message.

■ MSG_VIS_VUP_EC_ENSURE_OD_NOT_REFERENCED

```
void	MSG_VIS_VUP_EC_ENSURE_OD_NOT_REFERENCED(
	optr  object);
```

This message travels up the visible object tree, ensuring that the optr passed is not stored by any objects in the tree. This message may be sent when the object is being destroyed; any objects methods encountering a match between a stored optr and the passed optr should force a fatal error.

Source: Unrestricted.

Destination: Any visible object in the affected visible tree.

Parameters: *object* The handle of the object being destroyed.

Return: Nothing—if it returns, no error was encountered.

Interception: Do not intercept this message.

23.8 Creating Specific UIs

It is possible, using visible object classes, to create your own specific UIs. It is a complex and difficult task, however, and only very few developers will expect to do this. Documentation on creating a custom specific UI may appear either in a future release of this documentation or under separate cover. Contact Geoworks Developer Products for more information.

23.9 Basic Summary

Visible objects are involved in several basic operations, each of which has its own primary messages:

◆ Drawing
Vis objects must handle MSG_VIS_DRAW (page 1322) if they are to draw after their window has been exposed. Objects wishing to redraw themselves outside of exposure events can call MSG_VIS_VUP_CREATE_GSTATE (page 1319) to create a GState in their view windows for drawing.

◆ Being added to an object tree
When an object is added to a visible tree, it must then force a redraw of the tree. It can be added to the tree with MSG_VIS_ADD_CHILD (page 1377), and the tree can be updated with MSG_VIS_VUP_UPDATE_WIN_GROUP (page 1327).

◆ Being removed from an object tree
A visible object may be removed from a visible tree with MSG_VIS_REMOVE (page 1378). This message will close the object (take it off the screen) and then segregate it from its parent.

- Undergoing visual update
  The visual update mechanism is automatic for the most part. An object that requires visual update, however, can use MSG_VIS_MARK_INVALID (page 1325) in conjunction with MSG_VIS_VUP_UPDATE_WIN_GROUP (page 1327) to cause the update.

- Geometry management
  Geometry management of visible objects can be as simple as setting a few flags in the instance data or as complex as manually setting position and size of the objects. For a full description of geometry management of visible objects, see “Positioning Visible Objects” on page 1338.

- Being Destroyed
  Visible objects may be destroyed even when they are part of a visible tree that is on the screen. MSG_VIS_DESTROY (page 1374) first removes the object from the tree (with MSG_VIS_REMOVE) and then destroys it (through the **MetaClass** detach and destroy mechanisms).

VisComp

24

VisCompClass allows visible objects to have children. VisComp objects are the grouping elements within visible object trees.

If you need to have a visible object tree with several levels, or if you need to manage the geometry of visible objects, you will want to read this chapter. You should already be familiar with **VisClass** and with visible object trees. Both of these topics are discussed in “VisClass,” Chapter 23. If you have not yet read that chapter, you should do so now.

24.1 VisCompClass Features

Composite visible objects provide several features and functions that normal visible objects can not. **VisCompClass** has several instance data fields above and beyond those found in **VisClass** (since **VisCompClass** is a subclass of **VisClass**, it inherits all its instance data and methods). Although it only handles a few messages not defined in **VisClass**, it provides much more functionality. Some of the main features of composite objects are listed below:

- They can have children.
  Normal visible objects can exist only as leaves of a visible object tree. **VisCompClass** can have children, allowing the tree to be built to any number of levels. See section 23.5 of chapter 23 for full information on visible trees.

- They can manage their children’s geometry.
  A composite’s children can be managed arbitrarily by the composite, or the composite can use the geometry manager to automatically position and size its children. Since a composite can have other composites as its children, this sizing can descend recursively throughout the visible tree, making the entire tree’s geometry completely self-managed. See “Managing Geometry” on page 1417 for information on geometry management.

- They can create and manage their own windows.
  Composite objects can create their own graphics windows. Though this is not typically done (except in Specific UI libraries), it is possible. See

"Managing Graphic Windows" on page 1420 for more information on windows.

- They support other **VisClass** functions.
  VisCompClass inherits all the instance data fields and messages of **VisClass**. As such, it can do everything normal **VisClass** objects can do. See "VisClass," Chapter 23 for full information on **VisClass**.

24.2

24.2 VisCompClass Instance Data

As stated above, **VisCompClass** inherits all the instance data fields from **VisClass**. All of those fields may be set and reset as they could be for an object of that class. Composite objects also have four other instance fields, shown in Code Display 24-1.

Code Display 24-1 VisCompClass Instance Fields

```
/* The VisCompClass instance data fields are shown below and are discussed in
 * detail throughout the chapter. */

        /* VCI_comp
         * VCI_comp contains the link to the composite object's first child. */

    @instance @composite        VCI_comp = VI_link;

        /* VCI_gadgetExcl
         * VCI_gadgetExcl is an optr to the object that currently has the gadget
         * exclusive. This field is rarely used directly by applications. */

    @instance optr              VCI_gadgetExcl;

        /* VCI_window
         * VCI_window contains the window handle of the graphics window associated
         * with this object. This field is rarely accessed directly by
         * applications; it is set by the visual update mechanism. */

    @instance WindowHandle      VCI_window = NullHandle;

        /* VCI_geoAttrs
         * VCI_geoAttrs is a record that defines some of the geometry management
         * information for the composite. None of its values are set by default;
         * the possible flags in this record are shown after the definition. */
```

```
@instance VisCompGeoAttrs   VCI_geoAttrs = 0;
    /* Possible flags:
     *      VCGA_ORIENT_CHILDREN_VERTICALLY          0x80
     *      VCGA_INCLUDE_ENDS_IN_CHILD_SPACING       0x40
     *      VCGA_ALLOW_CHILDREN_TO_WRAP              0x20
     *      VCGA_ONE_PASS_OPTIMIZATION               0x10
     *      VCGA_CUSTOM_MANAGE_CHILDREN              0x08
     *      VCGA_HAS_MINIMUM_SIZE                    0x04
     *      VCGA_WRAP_AFTER_CHILD_COUNT              0x02
     *      VCGA_ONLY_DRAWS_IN_MARGINS               0x01
     */

    /* VCI_geoDimensionAttrs
     * VCI_geoDimensionAttrs is a record that contains additional information
     * about the composite’s geometry. This field contains two two-bit fields
     * among its other flags. The default settings are shown in the definition;
     * all possible flags and settings are shown following. */

@instance VisCompGeoDimensionAttrs   VCI_geoDimensionAttrs = 0;
    /* Possible flags:
     * Width Justification flags (mutually exclusive):
     * VCGDA_WIDTH_JUSTIFICATION                     0xc0
     *      WJ_LEFT_JUSTIFY_CHILDREN                 0x00
     *      WJ_RIGHT_JUSTIFY_CHILDREN                0x40
     *      WJ_CENTER_CHILDREN_HORIZONTALLY          0x80
     *      WJ_FULL_JUSTIFY_CHILDREN_HORIZONTALLY    0xc0
     *
     * Height Justification flags (mutually exclusive):
     * VCGDA_HEIGHT_JUSTIFICATION                    0x0c
     *      HJ_TOP_JUSTIFY_CHILDREN                  0x00
     *      HJ_BOTTOM_JUSTIFY_CHILDREN,              0x04
     *      HJ_CENTER_CHILDREN_VERTICALLY            0x08
     *      HJ_FULL_JUSTIFY_CHILDREN_VERTICALLY      0x0c
     *
     * Other flags:
     *      VCGDA_EXPAND_WIDTH_TO_FIT_PARENT         0x20
     *      VCGDA_DIVIDE_WIDTH_EQUALLY               0x10
     *      VCGDA_EXPAND_HEIGHT_TO_FIT_PARENT        0x02
     *      VCGDA_DIVIDE_HEIGHT_EQUALLY              0x01
     */

@default VI_typeFlags = VTF_IS_COMPOSITE;
```

24.2.1 VCI_comp

Normal visible objects have just one instance field for tree construction: the *VI_link* field that points to the object's next sibling. For an object to have children, it must also have an instance field declared with the keyword **@composite**; this field, in **VisCompClass**, is *VCI_comp*. The composite field contains the optr of the object's first child; the link field contains a pointer to either the object's next sibling or its parent.

Applications should never access the *VI_link* or *VCI_comp* fields directly; instead, these fields can be altered or queried with the following messages (all defined in **VisClass**):

MSG_VIS_ADD_CHILD
This message adds a new child object to the composite.

MSG_VIS_REMOVE_CHILD
This message removes a child from the composite.

MSG_VIS_MOVE_CHILD
This message moves a child within the composite's list of children.

MSG_VIS_GET_PARENT
This message returns a pointer to the object's parent.

Several other messages may also be used to change or query any visible object's *VI_link* and *VCI_comp* fields. These are discussed in "Working with Visible Object Trees" on page 1373 of "VisClass," Chapter 23.

24.2.2 VCI_gadgetExcl

Since the composite object must manage several other objects, it must also keep track of certain hierarchies used by the UI; the gadget exclusive is one of these. Within each branch of the visible object tree, only one visible object may have the gadget exclusive at a time. The gadget exclusive is kept track of via a path of pointers from the top of the tree down to the object having the exclusive.

The *VCI_gadgetExcl* field determines which child of the composite has the gadget exclusive. The child indicated in this field may or may not actually have the gadget exclusive for the entire visible tree; if the branch does not have the exclusive, neither will the child object. The gadget exclusive is similar to the other hierarchies of the input manager and acts the same way.

Applications never access the *VCI_gadgetExcl* field of a composite directly. Instead, this field may be set with the following three **VisClass** messages:

MSG_VIS_TAKE_GADGET_EXCL
This message causes a visible object to set itself as having the gadget exclusive. The object's parent composite will set its *VCI_gadgetExcl* field appropriately.

MSG_VIS_RELEASE_GADGET_EXCL
This message causes a visible object to relinquish its hold on the gadget exclusive. The object's parent composite will set its *VCI_gadgetExcl* field appropriately.

MSG_VIS_LOST_GADGET_EXCL
This message is sent to the object having the gadget exclusive for that branch when it has lost the exclusive.

The above messages are detailed in "The Gadget Exclusive and Focus Hierarchy" on page 1364 of "VisClass," Chapter 23.

24.2.3 VCI_window

Every visible object has a window associated with it in which it will be drawn. Normal **VisClass** objects have no control over what window they're associated with; they must be in the same window as their composite parents. Composites may need to appear in different windows from their parents, however. Thus, **VisCompClass** has the *VCI_window* field, which contains the window handle of the window in which the composite's branch will be drawn.

Applications will not access this field directly. Typically, composites will be drawn in the window associated with their parent composites or contents. Sometimes, however, an application will want a visible branch to appear in a window different from that of the rest of the visible tree. The composite then must use other **VisClass** messages for creating and associating a window. This process is discussed in more detail in section 24.3.2 on page 1420.

24.2.4 VCI_geoAttrs

The most-used feature of composite objects is their ability to manage the sizing and placement of their children. This is known as managing the composite's geometry.

The geometry management behavior implemented by a particular composite object is determined by its *VCI_geoAttrs* and *VCI_geoDimensionAttrs* fields. Both of these fields may be set and altered by applications as their geometry needs change. The *VCI_geoAttrs* field, specifically, determines the type of geometry management employed by the composite.

The *VCI_geoAttrs* field may contain any or all of the following eight flags:

VCGA_ORIENT_CHILDREN_VERTICALLY
This flag indicates that the composite's children should be arranged vertically rather than the default (horizontally).

VCGA_INCLUDE_ENDS_IN_CHILD_SPACING
When the composite is using full justification (see *VCI_geoDimensionAttrs*), this flag indicates that there should be as much space before the first child and after the last child as there is between the children. If this flag is not set, there will be no space outside the first and last children.

VCGA_ALLOW_CHILDREN_TO_WRAP
This flag will allow the children to wrap if their combined lengths won't allow them to fit within the composite's bounds. The composite will keep within the bounds of its parent, and its children will wrap as necessary. If this flag is not set, the composite will try to grow to be as large as necessary to fit all children.

VCGA_ONE_PASS_OPTIMIZATION
This flag makes the geometry manager make only one pass at managing the children. It should only be set if the children can be guaranteed not to wrap or resize.

VCGA_CUSTOM_MANAGE_CHILDREN
This flag indicates that the composite will manage its children without using the geometry manager. This allows the composite or its children to manually determine their positions and sizes. If this flag is set, the composite will keep its own

bounds in its *VI_bounds* field, just as other Vis objects. This flag in a composite indicates custom management for the composite's entire branch, not just its children.

VCGA_HAS_MINIMUM_SIZE
This flag indicates that the composite has a minimum size. The geometry manager will query the composite for this minimum and will ensure the object never gets smaller than that regardless of the size of its children. The minimum size must be returned by a custom MSG_VIS_COMP_GET_MINIMUM_SIZE handler.

VCGA_WRAP_AFTER_CHILD_COUNT
Used in conjunction with VCGA_ALLOW_CHILDREN_TO_WRAP, this flag will cause child wrapping after a certain number of children. This can cause wrapping based on the number of children rather than on child size. The geometry manager will query the composite with MSG_VIS_COMP_GET_WRAP_COUNT.

VCGA_ONLY_DRAWS_IN_MARGINS
This flag is used for optimized visual updates. It causes only the margins of the composite to be drawn when its image is marked invalid; all children of the composite must have their own images marked invalid if they are to be redrawn as well.

24.2.5 VCI_geoDimensionAttrs

The *VCI_geoDimensionAttrs* field determines how the composite manages its children in each dimension. It provides the geometry information not given in *VCI_geoAttrs* such as child justification and certain sizing behavior.

It contains three fields for each dimension (horizontal and vertical): The first field represents the justification of the children in that dimension. This field is two bits and can be one of four different enumerations. The second and third fields are sizing flags. These fields are listed below:

VCGDA_WIDTH_JUSTIFICATION
This is a two-bit field that can be set to any one of four possible width justifications. If the name of this field is used in place of one of the four values, full justification will be used. The justification can be set as a normal flag. The four different values are

WJ_LEFT_JUSTIFY_CHILDREN
Left justify the children.

WJ_RIGHT_JUSTIFY_CHILDREN
Right justify the children.

WJ_CENTER_CHILDREN_HORIZONTALLY
Center the children horizontally.

WJ_FULL_JUSTIFY_CHILDREN_HORIZONTALLY
Full justify the children. Horizontal full justification is only meaningful if the children are oriented horizontally (by clearing VCGA_ORIENT_CHILDREN_VERTICALLY in *VCI_geoAttrs*).

VCGDA_EXPAND_WIDTH_TO_FIT_PARENT
If this flag is set, the composite will try to expand to fill the available width of its parent. By default, this flag is not set; the composite will be only as wide as its children require.

VCGDA_DIVIDE_WIDTH_EQUALLY
If this flag is set, the composite will try to divide space equally between all its manageable, horizontally-oriented children. The composite will only suggest sizes—the children may or may not cooperate.

VCGDA_HEIGHT_JUSTIFICATION
This is a two-bit field that can be set to any one of four height justifications. If the name of this field is used in place of one of the four values, full justification will be used. The justification can be set as a normal flag. The four different values are

HJ_TOP_JUSTIFY_CHILDREN
Justify the children to the composite's top bound.

HJ_BOTTOM_JUSTIFY_CHILDREN
Justify the children to the composite's bottom bound.

HJ_CENTER_CHILDREN_VERTICALLY
Center the children vertically.

HJ_FULL_JUSTIFY_CHILDREN_VERTICALLY
Full justify the children. Vertical full justification is only meaningful if the children are oriented vertically (by setting VCGA_ORIENT_CHILDREN_VERTICALLY in *VCI_geoAttrs*).

Objects

VCGDA_EXPAND_HEIGHT_TO_FIT_PARENT
If this flag is set, the composite will try to expand to fill the available height of its parent. By default, this flag is not set; the composite will be only as tall as its children require.

VCGDA_DIVIDE_HEIGHT_EQUALLY
If this flag is set, the composite will try to divide space equally between all its manageable, vertically-oriented children. The composite will only suggest sizes—the children may or may not cooperate.

24.2.6 Managing Instance Data

MSG_VIS_COMP_GET_GEO_ATTRS, MSG_VIS_COMP_SET_GEO_ATTRS

To retrieve the flags currently set in both *VCI_geoAttrs* and *VCI_geoDimensionAttrs*, use MSG_VIS_COMP_GET_GEO_ATTRS. To set the attributes in either or both fields, use MSG_VIS_COMP_SET_GEO_ATTRS. both of these messages are detailed below.

■ MSG_VIS_COMP_SET_GEO_ATTRS

```
void    MSG_VIS_COMP_SET_GEO_ATTRS(
        word   attrsToSet,
        word   attrsToClear);
```

This message sets the flags in the composite object's *VCI_geoAttrs* and *VCI_geoDimensionAttrs* fields. The high byte of each parameter represents the dimension attributes, and the low byte represents the geometry attributes. This message does not invalidate or update the object's geometry.

Source: Unrestricted.

Destination: Any visible composite object.

Parameters: *attrsToSet* A word of attributes that should be set for the object. The high byte is a record of **VisCompGeoDimensionAttrs**, and the low byte is a record of **VisCompGeoAttrs**. The attributes set in this parameter will be set for the object.

attrsToClear A word of attributes to be cleared from the object's instance data. It has the same form as *attrsToSet*, above. Any attribute set in this parameter will be cleared in the instance fields.

Return: Nothing.

Interception: Unlikely.

■ MSG_VIS_COMP_GET_GEO_ATTRS

```
word	MSG_VIS_COMP_GET_GEO_ATTRS();
```

24.3

This message retrieves the flags set in the object's *VCI_geoAttrs* and *VCI_geoDimensionAttrs* fields. The high byte of the return value represents the dimension attributes, and the low byte represents the geometry attributes. This message does not invalidate or update the object's geometry.

Source: Unrestricted.

Destination: Any visible composite object.

Parameters: None.

Return: A word of flags. The high word is a record of type **VisCompGeoDimensionAttrs**; the low word is a record of type **VisCompGeoAttrs**. The high word represents the attributes set in the object's *VCI_geoDimensionAttrs* field, and the low byte represents the attributes set in the object's *VCI_geoAttrs* field.

Interception: Unlikely.

24.3 Using VisCompClass

With the exception of the geometry flags mentioned above and the management of children, the use of **VisCompClass** is little different from the use of **VisClass**. In fact, most of the functionality designed for **VisCompClass** is built directly into **VisClass**; for example, nearly all geometry management and object tree management messages are defined in **VisClass**. Some reminders are listed below, however.

Not all composite objects will have something to draw. Some composites will be used simply as grouping objects to manage visible children. In these cases, you will not have to subclass **VisCompClass** but can instead use the class directly.

Many composite objects will have something to draw, though. For example, a composite may want to draw a box around its children or wash a different background color behind them. If this is the case, the object must be a subclass of **VisCompClass** and must handle MSG_VIS_DRAW.

One of the primary functions of **VisCompClass** is to pass input events and other messages down and up the tree to the proper objects. This is done automatically. You can, however, change this behavior by subclassing **VisCompClass** and intercepting the messages in which the object will be interested.

24.3.1 Managing Geometry

```
MSG_VIS_COMP_GET_CHILD_SPACING,
MSG_VIS_COMP_GET_MINIMUM_SIZE, MSG_VIS_COMP_GET_MARGINS,
MSG_VIS_COMP_GET_WRAP_COUNT
```

A special feature of **VisCompClass**, and one that can be used in many ways, is its ability to automatically manage its children. By setting various flags in the composite's instance fields, you can have it control its children's sizing and position without additional code in your application.

Most of the flags you can set in *VCI_geoAttrs* and *VCI_geoDimensionAttrs* are explained fully in section 24.2 on page 1408. You should especially be aware that if you do not want to use the geometry management capabilities of **VisCompClass**, you should set VCGA_CUSTOM_MANAGE_CHILDREN in *VCI_geoAttrs*.

In addition to the messages provided in **VisClass** for geometry management, **VisCompClass** has four that return information about its current geometry. These are necessary because composites may be children of other composites, and therefore they may be managed. These messages are detailed below.

Most of the issues of geometry management are discussed in "Positioning Visible Objects" on page 1338 of "VisClass," Chapter 23.

■ DWORD_CHILD_SPACING

```
word      DWORD_CHILD_SPACING(val);
          SpacingAsDWord val;
```

This macro extracts the child spacing from the given **SpacingAsDWord** value. Use it with MSG_VIS_COMP_GET_CHILD_SPACING.

■ DWORD_WRAP_SPACING

```
word      DWORD_WRAP_SPACING(val);
          SpacingAsDWord val;
```

This macro extracts the wrap spacing from the given **SpacingAsDWord** value. Use it with MSG_VIS_COMP_GET_CHILD_SPACING.

■ MAKE_SPACING_DWORD

```
SpacingAsDWord MAKE_SPACING_DWORD(child, wrap);
          word   child;
          word   wrap;
```

This macro creates a **SpacingAsDWord** dword from the two given arguments. The *child* argument is the child spacing, and the *wrap* argument is the wrap spacing. Use this macro in your handler (if any) for the message MSG_VIS_COMP_GET_CHILD_SPACING.

■ MSG_VIS_COMP_GET_CHILD_SPACING

```
SpacingAsDWord MSG_VIS_COMP_GET_CHILD_SPACING();
```

This message returns the child spacing used by the composite. The high word of the return value is the spacing between lines of wrapped children; the low word is the horizontal spacing between children.

Source: Unrestricted.

Destination: Any visible composite object—typically sent by a composite to itself during geometry calculations.

Parameters: None.

Return: A dword containing the child spacing used by the composite. The dword contains two values: The child spacing—the amount of spacing placed between the composite's children—can be extracted from the return value with the macro DWORD_CHILD_SPACING. The wrap spacing—the amount of space placed between lines of wrapped children—can be extracted from the return value with the macro DWORD_WRAP_SPACING.

Interception: If a composite wants special child or wrap spacing other than the default, it should subclass this message and return the desired values. There is no need to call the superclass in the method.

Tips: In your handler, you can use the macro MAKE_SPACING_DWORD to form the return value from the two spacing values.

■ MSG_VIS_COMP_GET_MINIMUM_SIZE

```
SizeAsDWord MSG_VIS_COMP_GET_MINIMUM_SIZE();
```

This message returns the minimum size of the composite. It is used by the geometry manager if the composite has VCGDA_HAS_MINIMUM_SIZE set. This message does not invalidate or update the object's geometry.

Source: Unrestricted.

Destination: Any visible composite object—typically sent by a composite to itself during geometry calculations.

Parameters: None.

Return: A dword containing the minimum size of the composite. The high word is the width, and the low word is the height. Use the macros DWORD_WIDTH and DWORD_HEIGHT to extract the individual values from the **SizeAsDWord** structure.

Interception: Any composite that wants a minimum size should subclass this message and return its desired size. There is no need to call the superclass in your handler.

Tips: In your handler, use the macro MAKE_SIZE_DWORD to create the **SizeAsDWord** return value from the width and height. This macro has the same format as MAKE_SPACING_DWORD.

■ MSG_VIS_COMP_GET_MARGINS

```
void        MSG_VIS_COMP_GET_MARGINS(
            Rectangle *retValue);
```

This message returns the margins the composite should use when recalculating its child spacing. If you want a special left, top, right, or bottom margin around the composite's children, intercept this message and return the margin(s) in the appropriate field(s) of the **Rectangle** structure.

Source: Unrestricted.

Destination: Any visible composite object—typically sent by a composite to itself during geometry calculations.

Parameters: *retValue* A pointer to an empty **Rectangle** structure that will be filled with the composite's desired margins.

Return: The pointer to the filled Rectangle structure is preserved. The structure contains the four margins desired by the object outside of its bounds (e.g. if *retValue->R_top* is 100 upon return, the composite is requesting 100 points of extra "margin" spacing below its top bound before its children are placed).

Interception: Any composite that wants extra margin space added to its bounds when geometry is calculated should subclass this message and return its desired margins. There is no need to call the superclass in your handler.

■ MSG_VIS_COMP_GET_WRAP_COUNT

```
word    MSG_VIS_COMP_GET_WRAP_COUNT();
```

This message returns the number of children to be counted before wrapping if the composite has VCGA_WRAP_AFTER_CHILD_COUNT set.

Source: Unrestricted.

Destination: Any visible composite object—typically sent by a composite to itself during geometry calculations.

Parameters: None.

Return: The number of children that will be allowed before the composite wraps additional children to a new line.

Interception: Any composite that wants to wrap after a certain number of children should subclass this message and return the proper number of children. There is no need to call the superclass in your handler.

24.3.2 Managing Graphic Windows

It is very rare that a visible object will want to create its own window without using a generic object. This practice is highly discouraged as well because it will almost certainly violate some principles of most specific UI specifications.

You can, however, set up a visible object to have its own window with the following steps:

1 Set up the VisComp object as a window group.
Either instantiate a new VisComp object or set it up and load it in, off-screen. Then set the flags VTF_IS_WINDOW and VTF_IS_WIN_GROUP in the composite to make it a window and the top of the visible tree.

2 Add visible children to the window object.
Using MSG_VIS_ADD_CHILD, add any visible object children to the window group as you need. If some of the children had possibly been removed from the visible tree earlier, you may have to mark them invalid.

3 Add your window to the field window.
Determine the handle of the parent window you need, then set your window group object as a child of the parent.

4 Set the top object in your window group visible.
Set VA_VISIBLE using MSG_VIS_SET_ATTR to mark the window group visible. The window will be opened, and it will receive a MSG_META_EXPOSED indicating that it and all its children should draw themselves.

24.3

Objects

VisContent

25

VisContentClass provides objects that can be used to head visible trees and interact with the GenView. The VisContent is a necessary class for any application that plans on displaying visible objects in a view.

You should be familiar with visible objects, visible object trees, and the GenView class. These topics can be found in “VisClass,” Chapter 23 and “GenView,” Chapter 9.

Because **VisContentClass** is a subclass of **VisCompClass**, it inherits all the instance data fields and messages of both that class as well as **VisClass**. You will likely want to read about both of those classes before reading this chapter. You can just read section 25.2 on page 1440 if you need a basic VisContent object—that section describes the most basic ways to use the VisContent without understanding its inner workings.

One particular area of interest involving VisContent objects is when using the GenDocument and document control objects. A GenDocument at run-time gets resolved by the specific UI into some permutation of VisContent; thus, when using GenDocuments, you don’t use a VisContent as well. See “GenDocument,” Chapter 13 for more information on documents in a view.

25.1 VisContent Instance Data

The VisContent has many instance data fields above and beyond those defined in **VisClass** and **VisCompClass**. These new instance fields deal with such topics as content attributes, the way the content interacts with the view, the document described by the content, and how the content handles input.

All the instance fields defined by **VisContentClass** and their definitions are shown in Code Display 25-1. Each field is described later in this section.

Code Display 25-1 VisContent Instance Data Fields

```
/* All the instance fields defined in VisContentClass are described below. You
 * will likely use very few of them, if any; see the later sections of this
 * chapter for specific information about each field. */

    /* The following fields deal with the GenView and the view window. */
@instance optr              VCNI_view = 0;
@instance WindowHandle      VCNI_window = 0;
@instance word              VCNI_viewHeight = 0;
@instance word              VCNI_viewWidth = 0;

    /* VCNI_attrs determines the content’s attributes. */
@instance VisContentAttrs   VCNI_attrs = 0;

    /* The possible flags for VCNI_attrs are
     *      VCNA_SAME_WIDTH_AS_VIEW                         0x80
     *      VCNA_SAME_HEIGHT_AS_VIEW                        0x40
     *      VCNA_LARGE_DOCUMENT_MODEL                       0x20
     *      VCNA_WINDOW_COORDINATE_MOUSE_EVENTS             0x10
     *      VCNA_ACTIVE_MOUSE_GRAB_REQUIRES_LARGE_EVENTS    0x08
     *      VCNA_VIEW_DOC_BOUNDS_SET_MANUALLY               0x04     */

    /* The following fields determine features of the content’s document. */
@instance PointDWord        VCNI_docOrigin = {0, 0};
@instance PointWWFixed      VCNI_scaleFactor = {{0, 1}, {0, 1}};

    /* The following fields deal with how the content handles input. */
@instance ChunkHandle       VCNI_prePassiveMouseGrabList = 0;
@instance VisMouseGrab      VCNI_impliedMouseGrab =
                                {0, 0, {0, 0}, (VIFGF_MOUSE | VIFGF_PTR), 0};
@instance VisMouseGrab      VCNI_activeMouseGrab =
                                {0, 0, {0, 0}, 0, 0};
@instance ChunkHandle       VCNI_postPassiveMouseGrabList = 0;
@instance KbdGrab           VCNI_kbdGrab = {0, 0};
@instance FTVMCGrab         VCNI_focusExcl = {0, MAEF_FOCUS};
@instance FTVMCGrab         VCNI_targetExcl = {0, MAEF_TARGET};
@instance Handle            VCNI_holdUpInputQueue = 0;
@instance word              VCNI_holdUpInputCount = 0;
@instance byte              VCNI_holdUpInputFlags = 0;

    /* The type flags of the content are special and should not be altered. */
@default    VI_typeFlags =  VTF_IS_COMPOSITE | VTF_IS_WINDOW |
                            VTF_IS_CONTENT | VTF_IS_WIN_GROUP |
                            VTF_IS_INPUT_NODE;
```

Objects

25.1.1 The VCNI_attrs Field

VCNI_attrs, MSG_VIS_CONTENT_SET_ATTRS,
MSG_VIS_CONTENT_GET_ATTRS

The *VCNI_attrs* field is a record of **VisContentAttrs** that contains several attributes which affect how the content object interacts with the view and with the visible object tree. You can set these attributes with MSG_VIS_CONTENT_SET_ATTRS dynamically, and you can retrieve them with MSG_VIS_CONTENT_GET_ATTRS.

The flags in this field are shown below. None of them is set by default.

VCNA_SAME_WIDTH_AS_VIEW
This flag indicates that the content's width should follow the width of the view window, if possible. Most likely, you will want to set this if the view is not horizontally scrollable.

VCNA_SAME_HEIGHT_AS_VIEW
This flag indicates that the content's height should follow the height of the view window, if possible. Most likely you will want to set this flag if the view is not vertically scrollable.

VCNA_LARGE_DOCUMENT_MODEL
This flag indicates that the content object manages a large document (32-bit coordinates rather than the standard 16-bit coordinates). For information on how this affects the content and its children, see section 23.6.1 on page 1395.

VCNA_WINDOW_COORDINATE_MOUSE_EVENTS
This flag must be set if VCNA_LARGE_DOCUMENT_MODEL is set. It indicates that the associated GenView will pass input events with window coordinates rather than document coordinates. The default handlers in the content object will then automatically translate the events into document coordinates.

VCNA_ACTIVE_MOUSE_GRAB_REQUIRES_LARGE_EVENTS
This flag indicates that the object that currently has the active mouse grab requires mouse input events to carry large document coordinates rather than the standard document coordinates. This flag is set and reset with the message MSG_VIS_VUP_ALTER_INPUT_FLOW. Large content objects should not set this flag in their Goc declarations.

VCNA_VIEW_DOC_BOUNDS_SET_MANUALLY
Not often used, this flag indicates that the content should not send its document bounds off to the view during a geometry update. The GenView's document bounds must be set manually, most likely with the GenView message MSG_GEN_VIEW_SET_DOC_BOUNDS.

VCNA_VIEW_DOES_NOT_WIN_SCROLL
This flag indicates that the view does not scroll but instead sends MSG_META_CONTENT_VIEW_ORIGIN_CHANGED to the content when the user interacts with the scroller. The UI will use this flag to invalidate the correct region of the content. This should be set when ATTR_GEN_VIEW_DO_NOT_WIN_SCROLL is set in the GenView. See "Scrolling" on page 543 of "GenView," Chapter 9, for full information on view scrolling.

■ MSG_VIS_CONTENT_SET_ATTRS

```
void        MSG_VIS_CONTENT_SET_ATTRS(
            VisContentAttrs attrsToSet,
            VisContentAttrs attrsToClear);
```

This message sets the *VCNI_attrs* field of the content object according to the passed values.

Source: Unrestricted.

Destination: Any VisContent object.

Parameters: *attrsToSet* This is a record of **VisContentAttrs** to set. A flag set in this field will be set in the object's *VCNI_attrs* field.

attrsToClear This is a record of **VisContentAttrs** to clear. A flag set in this field will be cleared in the object's *VCNI_attrs* field. This parameter takes precedence over *attrsToSet*; that is, if a flag is set in both parameters, it will end up cleared.

Return: Nothing.

Interception: Unlikely.

■ MSG_VIS_CONTENT_GET_ATTRS

```
VisContentAttrs MSG_VIS_CONTENT_GET_ATTRS();
```

This message returns the current contents of the object's *VCNI_attrs* field, a record of **VisContentAttrs**.

Source: Unrestricted.

Destination: Any VisContent object.

Parameters: None.

Return: A record of **VisContentAttrs** reflecting the flags currently set in the object's *VCNI_attrs* field.

Interception: Unlikely.

25.1.2 Fields That Affect the View

VCNI_view, VCNI_viewHeight, VCNI_viewWidth, VCNI_window, MSG_VIS_CONTENT_GET_WIN_SIZE

Because the content must interact directly with the view, it must maintain some information about both the GenView object and its associated window. **VisContentClass** defines four fields that deal exclusively with view-related information. Each of these fields is described below.

VCNI_view This field contains the optr of the GenView object. The content should not ever actually use this field for the view's optr; if you need to contact the view, you should instead use the messages MSG_VIS_VUP_SEND_TO_OBJECT_OF_CLASS and MSG_VIS_VUP_CALL_OBJECT_OF_CLASS. The field will be set automatically by the GenView when the view is opened; the view will send a MSG_META_CONTENT_VIEW_OPENING.

VCNI_viewHeight
This field contains the height, in document coordinates, of the view window. This will be set by the view object and should not be altered by the application; it can be retrieved with the message MSG_VIS_CONTENT_GET_WIN_SIZE (below). The view automatically notifies the content each time the window size changes with MSG_VIS_CONTENT_VIEW_SIZE_CHANGED.

VCNI_viewWidth

This field contains the width, in document coordinates, of the view window. This will be set by the view object and should not be altered by the application; it can be retrieved with the message MSG_VIS_CONTENT_GET_WIN_SIZE (below). The view automatically notifies the content on each window resize with MSG_META_CONTENT_VIEW_SIZE_CHANGED.

VCNI_window

This field contains the window handle of the view's window. This field is set by the view object and should not be accessed by the application. If an object under the content needs to get the window handle, it should use MSG_VIS_QUERY_WINDOW. This field will be set automatically by the GenView when the view's window is first opened. The view will send the messages MSG_META_CONTENT_VIEW_WIN_OPENED and MSG_META_CONTENT_VIEW_OPENING to set the window handle. When the view window closes, the view will send MSG_META_CONTENT_VIEW_CLOSING and MSG_META_CONTENT_VIEW_WIN_CLOSED.

The messages shown above that are sent by the view to the content are detailed in "Messages Received from the View" on page 1443.

■ MSG_VIS_CONTENT_GET_WIN_SIZE

```
SizeAsDWord MSG_VIS_CONTENT_GET_WIN_SIZE();
```

This message returns the size of the content object's associated window in terms of width and height.

Source: Unrestricted.

Destination: Any VisContent object.

Parameters: None.

Return: A **SizeAsDWord** value with the window's width in the high word and the window's height in the low word. Use the DWORD_HEIGHT and DWORD_WIDTH macros to extract the proper values.

Interception: Unlikely.

25.1.3 Fields That Affect the Document

VCNI_docOrigin, VCNI_scaleFactor,
MSG_VIS_CONTENT_SET_DOC_BOUNDS,
MSG_VIS_CONTENT_RECALC_SIZE_BASED_ON_VIEW

The GenView object maintains quite a bit of information about the document as managed by the content object. The content must also keep information about the document and how the view is displaying it. This information is stored in two fields, *VCNI_docOrigin* and *VCNI_scaleFactor*, both detailed below.

The document bounds of the content are typically equal to the bounds of the VisContent itself. The content's bounds are stored in the *VI_bounds* field inherited from **VisClass**. When a content is managing layers or large documents, however, its bounds are set to zero. It then manages its document bounds within the layer objects and the GenView. When the document bounds change, a MSG_VIS_CONTENT_SET_DOC_BOUNDS should be sent to the content to get it to notify all its layer children and the GenView of the new bounds. This message is shown at the end of this section.

If the content is not a large document and is set up to follow the GenView's geometry (it has either or both of VCNA_SAME_WIDTH_AS_VIEW or VCNA_SAME_HEIGHT_AS_VIEW set), it will be affected by changes in the view's geometry. During geometry updates, the view will send it a MSG_VIS_CONTENT_RECALC_SIZE_BASED_ON_VIEW. This message sets the content's width and/or height and therefore affects its *VI_bounds* field. This message is also shown at the end of this section.

VCNI_docOrigin

This field contains the current origin of the view window. The origin is the location of the view's upper left corner in the document (where the scrollers are). This field is of type **PointDWord**, which has the following structure:

```
typedef struct {
    sdword   PD_x;    /* x coordinate of origin */
    sdword   PD_y;    /* y coordinate of origin */
} PointDWord;
```

Normally, this field is set when the view is first opened or when the view is scrolled, scaled, or otherwise changed in document position. The view will send the message MSG_META_CONTENT_VIEW_ORIGIN_CHANGED to indicate the origin is different from its current setting.

VCNI_scaleFactor

This field contains the current scale factor the view is displaying. Scaling is implemented almost entirely in the GenView object; some content objects, however, will want to react in a special way when the scale factor is changed. The scale factor is stored in a **PointWWFixed** structure, as follows:

```
typedef struct {
    WWFixed    PF_x;   /* horizontal scale factor */
    WWFixed    PF_y;   /* vertical scale factor */
} PointWWFixed;
```

The **WWFixed** structures that determine the scale factor in each dimension consist of two elements. This structure is shown below:

```
typedef struct {
    word    WWF_frac;   /* fractional portion */
    word    WWF_int;    /* integral portion */
} WWFixed;
```

The *VCNI_scaleFactor* field in the content is never set directly by the application; instead, it is set with MSG_META_CONTENT_VIEW_SCALE_FACTOR_CHANGED whenever the view's scale factor changes.

Messages that set these fields are discussed in section 25.2.2 on page 1443.

■ MSG_VIS_CONTENT_SET_DOC_BOUNDS

```
void    MSG_VIS_CONTENT_SET_DOC_BOUNDS(@stack
        sdword bottom,
        sdword right,
        sdword top,
        sdword left);
```

This message is used to set the document bounds of a content's large document. The content will send MSG_GEN_VIEW_SET_DOC_BOUNDS to its view and MSG_VIS_LAYER_SET_DOC_BOUNDS to each of its children with the new bounds. The recipient content *must* have the flag VCNA_LARGE_DOCUMENT_MODEL set in *VCNI_attrs*.

Source: Unrestricted.

Destination: Any *large* VisContent object—if the content is not using the large document model, an error will result.

Parameters: *bottom, right, top, left*
The new document bounds in 32-bit document coordinates.

Return: Nothing.

Interception: Unlikely.

Warnings: You may only send this message to a content that is using the large document model. All other contents will not handle this message but will result in a fatal error.

■ MSG_VIS_CONTENT_RECALC_SIZE_BASED_ON_VIEW

```
SizeAsDWord MSG_VIS_CONTENT_RECALC_SIZE_BASED_ON_VIEW();
```

This message causes the content to recalculate its size based on the view's geometry. It will try to set its width if it has VCNA_SAME_WIDTH_AS_VIEW set, and it will try to set its height if it has VCNA_SAME_HEIGHT_AS_VIEW set.

Source: Unrestricted—typically sent by the view or by the content to itself during geometry updates.

Destination: Any VisContent object.

Parameters: None.

Return: A **SizeAsDWord** value indicating the new size of the content. Use the macros DWORD_HEIGHT and DWORD_WIDTH to extract the appropriate values.

Interception: Unlikely.

25.1.4 Fields That Affect Input Events

One of the main features of **VisContentClass** is its ability to handle, manage, and pass on input events sent through the GenView. The content has a large amount of functionality built into it to provide these features.

25.1.4.1 Mouse and Keyboard Grabs

VCNI_prePassiveMouseGrabList, VCNI_impliedMouseGrab, VCNI_activeMouseGrab, VCNI_postPassiveMouseGrabList, VCNI_kbdGrab, MSG_VIS_CONTENT_UNWANTED_MOUSE_EVENT, MSG_VIS_CONTENT_UNWANTED_KBD_EVENT, MSG_VIS_CONTENT_TEST_IF_ACTIVE_OR_IMPLIED_WIN

VisContentClass, as the head of the visible tree displayed in the view, keeps track of which object in its tree has each different type of input grab. With this information, the content can simply pass the input event directly to the object that has the grab. It can also easily send the event to both the prepassive and postpassive grab objects, if any.

To do this, it uses five instance fields that applications can not directly access. These fields are altered by the messages that allow an object to gain and release the subject grab. (These messages are all detailed in "Handling Input" on page 1351 of "VisClass," Chapter 23.) The instance fields are listed below:

VCNI_prePassiveMouseGrabList
This field is a pointer to a chunkarray containing the list of objects that currently have the prepassive mouse grab.

VCNI_impliedMouseGrab
This field is a **VisMouseGrab** (described below) structure that contains details about the object which has the implied mouse grab.

VCNI_activeMouseGrab
This field is a **VisMouseGrab** (described below) structure that contains details about the object which has the active mouse grab.

VCNI_postPassiveMouseGrabList
This field is a pointer to a chunkarray containing the list of objects that currently have the postpassive mouse grab.

VCNI_kbdGrab
This field is a **KbdGrab** structure that contains details about the object which currently has the keyboard grab.

You will probably not ever have to know the structure of these fields, how to set them, or the information they contain. However the two structures

VisMouseGrab and **KbdGrab** are shown in Code Display 25-2 for your information.

Code Display 25-2 Grab Data Structures

```
/* These structures are obscure, and you will likely never have to use them. */

/* The VisMouseGrab structure contains information about the object that
 * currently has the mouse grab. */

typedef struct {
    optr        VMG_object;     /* The optr of the object that has the grab.
                                 * If no object has the grab, this is zero. */
    WindowHandleVMG_gWin;       /* The window handle of the window with the object
                                 * having the grab, if it’s different from
                                 * the content’s window. If it’s in the content’s
                                 * window, this field contains zero.*/
    PointDWord  VMG_translation;/* The 32-bit translation applied to mouse events
                                 * if the large document model is in use. This is
                                 * set with a previous message call. */
    VisInputFlowGrabFlags VMG_flags; /* A record of VisInputFlowGrabFlags,
                                   * described below. */
    byte        VMG_unused;     /* Reserved byte. */
} VisMouseGrab;

/* The VisInputFlowGrabFlags determine the type and context of the grab. These
 * flags are not listed here for simplicity. You do not have to know these flags;
 * they are set with MSG_VIS_VUP_ALTER_INPUT_FLOW. */

/* The KbdGrab structure contains information about the object that currently
 * has the keyboard grab. */

typedef struct {
    optr        KG_OD;          /* The optr of the object that has the
                                 * keyboard grab. */
    word        KG_unused;      /* Reserved word. */
} KbdGrab;
```

In addition, the VisContent has the following messages that are affected by the input grab fields:

MSG_VIS_CONTENT_TEST_IF_ACTIVE_OR_IMPLIED_WIN

This message returns *true* if the passed window handle is the same as the active or implied window currently translating mouse events. It returns *false* otherwise. This is used by specific UI objects.

MSG_VIS_CONTENT_UNWANTED_MOUSE_EVENT
This message is sent by the content to itself if a mouse event has arrived and there is no implied or active grab in the content's visible tree. The default reaction is to beep if the event is a button press. It's highly unlikely that you'll send or intercept this message.

MSG_VIS_CONTENT_UNWANTED_KBD_EVENT
This message is sent by the content to itself if a keyboard event has arrived and there is no keyboard grab set up. It is highly unlikely that you will ever send or intercept this message.

■ MSG_VIS_CONTENT_TEST_IF_ACTIVE_OR_IMPLIED_WIN

```
Boolean  MSG_VIS_CONTENT_TEST_IF_ACTIVE_OR_IMPLIED_WIN(
         WindowHandle window);
```

This message checks to see if the passed window handle is the same as the window of the object having either the implied or active mouse grab. This is typically used by objects in a Specific UI library to determine if the mouse event was actually within the window or directly on the window's border.

Source: Unrestricted.

Destination: Any VisContent object.

Parameters: *window* The window handle to be checked.

Return: *True* if the window handle is the same as either the active or the implied window, *false* otherwise.

Interception: Unlikely.

■ MSG_VIS_CONTENT_UNWANTED_MOUSE_EVENT

```
void     MSG_VIS_CONTENT_UNWANTED_MOUSE_EVENT(
         VisMouseGrab *mouseGrab,
         word   inputState);
```

This message is received by the content if a mouse event was received and there was no active or implied grab. This is most frequently encountered when the user presses a mouse button outside a modal dialog box. The default action of this handler is to beep (on presses only, not releases) and get rid of the event as if it had been handled.

Source: Input flow mechanism.

Destination: The affected VisContent object.

Parameters: *mouseGrab* A pointer to the appropriate **VisMouseGrab** structure.

inputState Same as passed with the actual event.

Return: Nothing.

Interception: Unlikely.

■ MSG_VIS_CONTENT_UNWANTED_KBD_EVENT

```
void      MSG_VIS_CONTENT_UNWANTED_KBD_EVENT(
          word   character,
          word   flags,
          word   state);
```

This message will be received by the content if a keyboard event was sent and there was no keyboard grab set up. The default action of the content is to beep (on presses only, not releases) and return as if the event had been processed.

Source: Input flow mechanism.

Destination: The affected VisContent object.

Parameters: *character* The keyboard character pressed.

flags A word of flags: The low byte is a **CharFlags** record, and the high byte is a **ShiftState** record. Both are the same as passed with the original MSG_META_KBD_CHAR.

state A word containing two values: The low byte is a record of **ToggleState**, and the high byte is the scan code. Both are the same as passed with the original MSG_META_KBD_CHAR.

Return: Nothing.

Interception: Unlikely.

25.1.4.2 Focus and Target

`VCNI_focusExcl, VCNI_targetExcl, MSG_META_CONTENT_APPLY_DEFAULT_FOCUS`

In addition to keeping track of which of its children have the mouse and keyboard grabs, the content also keeps track of which objects have the focus and target input exclusives. Both *VCNI_focusExcl* and *VCNI_targetExcl* contain a structure of type **FTVMCGrab** that describes the object that has the subject exclusive. The messages sent by the GenView that set these fields are described in “Messages Received from the View” on page 1443.

25.1.4.3 Input Flow Control

```
VCNI_holdUpInputQueue, VCNI_holdUpInputCount,
VCNI_holdUpInputFlags,
MSG_VIS_CONTENT_HOLD_UP_INPUT_FLOW,
MSG_VIS_CONTENT_RESUME_INPUT_FLOW,
MSG_VIS_CONTENT_DISABLE_HOLD_UP,
MSG_VIS_CONTENT_ENABLE_HOLD_UP
```

GEOS allows a visible tree to hold up input—that is, input will be stored elsewhere while the visible tree is doing something else. This can be useful if complex tree operations are going on and you don’t want input to go to the wrong object.

VisContentClass provides three instance fields that define the input holdup mechanism. These three fields are

VCNI_holdUpInputQueue
This field contains the queue handle of the queue where held-up input will be temporarily stored. Input events will go into this event queue until they are allowed to be handled again; then they will be sent to their proper recipients.

VCNI_holdUpInputCount
This field contains a count of the number of objects that have requested that input be held up. If this count is positive, input will be held up and input events will go into the hold-up queue.

VCNI_holdUpInputFlags
This field contains a record of flags which determine the state of the hold-up mechanism. The following two flags are allowed:

HUIF_FLUSHING_QUEUE
This flag indicates that the hold-up queue is currently being flushed.

HUIF_HOLD_UP_MODE_DISABLED
This flag forces input events to flow normally. It is used

primarily by GEOS to ensure that the user can interact with a system-modal dialog box.

VisContentClass has four messages that it sends to itself to set the state of information hold-up. These messages are detailed below.

■ MSG_VIS_CONTENT_HOLD_UP_INPUT_FLOW

```
void    MSG_VIS_CONTENT_HOLD_UP_INPUT_FLOW();
```

This message increments the count in *VCNI_holdUpInputCount*. If this count is nonzero and HUIF_HOLD_UP_MODE_DISABLED is clear, subsequent input events will be sent into the hold-up queue until either the flag is set or the count once more drops to zero. 25.1

Source: Unrestricted.

Destination: Any VisContent object.

Interception: Unlikely.

Warnings: Do not forget to resume input with a later use of the message MSG_VIS_CONTENT_RESUME_INPUT_FLOW.

■ MSG_VIS_CONTENT_RESUME_INPUT_FLOW

```
void    MSG_VIS_CONTENT_RESUME_INPUT_FLOW();
```

This message decrements the count in *VCNI_holdUpInputCount*. If the count becomes zero with this call, the hold-up event queue is flushed and all the events in it are “played back.” If the count goes below zero, GEOS will give an error. Therefore, do not use this message unless it is preceded with a MSG_VIS_CONTENT_HOLD_UP_INPUT_FLOW.

Source: Unrestricted.

Destination: Any VisContent object.

Interception: Unlikely.

Warnings: If this message is used without first holding up input with MSG_VIS_CONTENT_HOLD_UP_INPUT_FLOW, an error will be the likely result. The error condition is the *VCNI_holdUpInputCount* field going below zero.

■ MSG_VIS_CONTENT_DISABLE_HOLD_UP

```
void    MSG_VIS_CONTENT_DISABLE_HOLD_UP();
```

This message sets the HUIF_HOLD_UP_MODE_DISABLED flag, forcing all input events to flow normally until the flag is cleared. In essence, it turns off the hold-up mechanism.

Source: Unrestricted.

Destination: Any VisContent object.

Interception: Unlikely.

■ MSG_VIS_CONTENT_ENABLE_HOLD_UP

```
void    MSG_VIS_CONTENT_ENABLE_HOLD_UP();
```

This message clears the HUIF_HOLD_UP_MODE_DISABLED flag, allowing input events to be held up in the hold-up event queue.

Source: Unrestricted.

Destination: Any VisContent object.

Interception: Unlikely.

25.2 Basic VisContent Usage

You will probably not have to subclass **VisContentClass** unless you are using a large document or you want the content to draw something in response to a MSG_VIS_DRAW. In most other cases, you can declare your content object directly on **VisContentClass**.

Most VisContent behavior is inherited directly from **VisCompClass**. Some additional behavior is detailed in the following sections.

If you are planning on using the document control objects with GenViews, you will not need VisContent objects. The GenDocument resolves at run-time into a subclass of VisContent and therefore replaces the need for an explicitly-defined VisContent object. See “GenDocument,” Chapter 13 for full information on GenDocuments.

25.2.1 Setting Up Sizing Behavior

Most geometry behavior revolves around whether the GenView is scrollable or not. You can set up several different sizing behaviors depending on the attributes you set in the GenView. Some are listed below with the attributes you should set in both the GenView and the VisContent.

Typically, setting the view's attributes will involve either setting or not setting the GVDA_SCROLLABLE flag in the appropriate GenView dimension; also, you will want to be aware of the GVDA_NO_LARGER_THAN_CONTENT and GVDA_NO_SMALLER_THAN_CONTENT flags.

25.2.1.1 If the Content Is Fixed Size

Typically, if the content object is a fixed size, the view will either conform to the content's size or scroll in one or both dimensions. This behavior is determined entirely within the GenView's instance data. Three types of this behavior are shown in Code Display 25-3.

Code Display 25-3 Sizing the View with a Fixed Content

```
/* This code display shows three different types of sizing behavior of the GenView
 * if its VisContent object is of a fixed size. Note that if the content is
 * managing its geometry, its bounds (and therefore the view's) will be determined
 * by the content's children. */

/* The view window is scrollable in both dimensions. This will result in the view
 * being sizable and scrollable in both dimensions. */

@object GenViewClass MyView = {
    GVI_content = @MyVisContent;
    GVI_horizAttrs = @default | GVDA_SCROLLABLE;
    GVI_vertAttrs = @default | GVDA_SCROLLABLE;
};

/* The view window is scrollable in only the vertical dimension. It follows the
 * width of the VisContent object and therefore does not scroll vertically. The
 * VisContent's VI_bounds field should be set by the content. */
```

```
@object GenViewClass MyView = {
    GVI_content = @MyVisContent;
    GVDI_horizAttrs = @default   | GVDA_NO_LARGER_THAN_CONTENT
                                 | GVDA_NO_SMALLER_THAN_CONTENT;
    GVDI_vertAttrs = @default    | GVDA_SCROLLABLE;
};

/* The view window sizes itself exactly to the size of the VisContent's bounds.
 * The VisContent's VI_bounds field should be set appropriately by the content.
 * Note that this is not a valid combination for VisContents that display large
 * documents or layer objects. */

@object GenViewClass MyView = {
    GVI_content = @MyVisContent;
    GVDI_horizAttrs = @default   | GVDA_NO_LARGER_THAN_CONTENT
                                 | GVDA_NO_SMALLER_THAN_CONTENT;
    GVDI_vertAttrs = @default    | GVDA_NO_LARGER_THAN_CONTENT
                                 | GVDA_NO_SMALLER_THAN_CONTENT;
}
```

Another type of behavior with fixed-size contents is called "keeping the aspect ratio." The view and content can work together to allow the user to resize the view while automatically setting the view's scale factor to keep the entire content in the window. This might be used, for example, for a game board; the game could be resized, and the entire game board would stay in the view.

In this situation, the content calculates its new size based on one of the view's dimensions. For example, when the user resizes the view, the content may keep the width but calculate the height based on the width. The proper scale factor is then set, and the content does not have to do anything special beyond that. To gain this behavior, set up your view and content as shown in Code Display 25-4.

Code Display 25-4 Keeping the View Aspect Ratio

```
/* This example shows a view and its content. The content object is of a fixed
 * size, and the view is resizable. The content/view pair will keep the aspect
 * ratio to automatically figure the view's height based on its width and then
 * scale the image to keep the entire bounds of the content within the view
 * window. */
```

```
@object GenViewClass MyView = {
    GVI_content = @MyVisContent;
    GVDI_horizAttrs = @default   | GVDA_NO_LARGER_THAN_CONTENT
                                 | GVDA_NO_SMALLER_THAN_CONTENT;
    GVDI_vertAttrs = @default    | GVDA_KEEP_ASPECT_RATIO;
};

@object VisContentClass MyVisContent = {
    VI_bounds = {0,     /* left bound */
                 0,     /* top bound */
                 250,   /* right bound */
                 250};  /* bottom bound */
    VCI_comp =          /* put any children here */;
    VCI_geoAttrs = VCGA_CUSTOM_MANAGE_CHILDREN;
                /* This is set because typically a content’s bounds are determined
                 * by its children. If we want to set our own bounds, we should
                 * custom manage our geometry. This is true of contents used with
                 * the views in the previous example. */
};
```

25.2.1.2 If the Content Is Variable Size

Many visible trees will have contents that are of variable size. How the content determines its size differs from use to use; some contents will adjust their geometries to those of their view objects, and some will resize themselves based on the geometry of their children.

Typically, if the content resizes itself based on its children’s geometry, the view either will be scrollable or will adjust its size to that of its content.

If the content rearranges its children to meet the size of the view, the view will not be scrollable and will not adjust its size to the content at all. Instead, the content will have the VCNA_SAME_WIDTH_AS_VIEW and VCNA_SAME_HEIGHT_AS_VIEW flags set in its *VCNI_attrs* field.

25.2.2 Messages Received from the View

MSG_META_CONTENT_SET_VIEW,
MSG_META_CONTENT_VIEW_ORIGIN_CHANGED,

```
MSG_META_CONTENT_VIEW_SCALE_FACTOR_CHANGED,
MSG_META_CONTENT_VIEW_OPENING,
MSG_META_CONTENT_VIEW_WIN_OPENED,
MSG_META_CONTENT_VIEW_SIZE_CHANGED,
MSG_META_CONTENT_VIEW_CLOSING,
MSG_META_CONTENT_VIEW_WIN_CLOSED
```

25.2

As detailed in the discussions on GenView, the view sends a sequence of messages to its content when the view is first opening and when it is closing. These messages set up the content's visible instance data and prime the visible tree to be drawn on the screen. The messages are handled by the default handlers in **VisContentClass**, and you do not need to add anything to them to make them work.

The messages sent to the content when the view is first created are

1 MSG_META_CONTENT_SET_VIEW
This message passes the view's optr to the content, setting the *VCNI_view* field properly.

2 MSG_META_CONTENT_VIEW_ORIGIN_CHANGED
This message passes the view's initial origin (which may be set other than the default) to the content.

3 MSG_META_CONTENT_VIEW_SCALE_FACTOR_CHANGED
This message passes the view's initial scale factor (which may be set other than the default) to the content.

4 MSG_META_CONTENT_VIEW_OPENING
This message is sent to the content when the view receives a MSG_VIS_OPEN. This notifies the content that the view is about to be put on the screen and that it should prepare itself to be drawn.

5 MSG_META_CONTENT_VIEW_WIN_OPENED
This message passes the window handle of the newly created view window so the content can record it in *VCNI_window*.

6 MSG_META_CONTENT_VIEW_SIZE_CHANGED
This message passes the view window's size (height and width) to the content so the content can determine its proper size and geometry.

7 MSG_META_EXPOSED
This message is sent when the view's window is finally on the screen. It

signifies that the content should draw itself and then send MSG_VIS_DRAWs to all its children.

The view will also send certain messages to the content when different things happen to change the content's instance data:

- MSG_META_CONTENT_VIEW_SIZE_CHANGED
  This message is passed whenever the view window's size changes for any reason.

- MSG_META_CONTENT_VIEW_SCALE_FACTOR_CHANGED
  This message is passed whenever the view's scale factor is changed, usually due to the user setting it from a menu.
- MSG_META_CONTENT_VIEW_ORIGIN_CHANGED
  This message is passed whenever the view's origin is changed, usually when the view is scrolled.
- MSG_META_EXPOSED
  This message is passed to the content whenever a portion of the view window becomes exposed and must be drawn. The content automatically translates this into a MSG_VIS_DRAW.

When the view is shutting down, it will send the following three messages to the content to set the proper data:

1. MSG_META_CONTENT_VIEW_CLOSING
   This message is sent to the content when the view receives a MSG_VIS_CLOSE. It indicates that the view is being taken off the screen and that all the visible objects in the content's tree should remove themselves from the screen.
2. MSG_META_CONTENT_VIEW_WIN_CLOSED
   This message is sent when the view's window is finally destroyed. The copy of the window handle in *VCNI_window* will be thrown out now so no drawing to the stale window handle will be done.
3. MSG_META_CONTENT_SET_VIEW
   This message is sent to set the content's *VCNI_view* field to a null handle. When the view is finally taken off the screen, it no longer should have a content associated with it since it is meaningless to work with a visible tree that is not on the screen. If the view is opened again later, the content will receive another MSG_META_CONTENT_VIEW_OPENING and will once again be passed the view's optr.

■ MSG_META_CONTENT_SET_VIEW

```
void      MSG_META_CONTENT_SET_VIEW(
          optr   view);
```

This message passes the optr of the GenView object that will display this content object. The default handler will set the content's *VCNI_view* field to the passed optr. This message is also used when the view has been shut down; the passed optr will be null.

Source: Unrestricted—typically sent by a GenView to its content object.

Destination: Any VisContent or Process object acting as the content of a GenView.

Parameters: *view* The optr of the GenView using this object as its content.

Return: Nothing.

Interception: Unlikely.

■ MSG_META_CONTENT_VIEW_ORIGIN_CHANGED

```
void      MSG_META_CONTENT_VIEW_ORIGIN_CHANGED(@stack
          WindowHandle viewWindow,
          sdword             xOrigin,
          sdword             yOrigin);
```

This message notifies the content that the view's origin has changed. The content will set its *VCNI_docOrigin* field to the passed values.

Source: Unrestricted—typically sent by a GenView to its content object.

Destination: Any VisContent or Process object acting as the content of a GenView.

Parameters:

viewWindow	The window handle of the GenView's window.
xOrigin	The new horizontal origin of the view.
yOrigin	The new vertical origin of the view.

Return: Nothing.

Interception: Any content that is managing large documents will probably need to subclass this message and apply the proper translations for the 32-bit coordinates.

■ MSG_META_CONTENT_VIEW_SCALE_FACTOR_CHANGED

```
void    MSG_META_CONTENT_VIEW_SCALE_FACTOR_CHANGED(@stack
        WindowHandle viewWindow,
        WWFixedAsDWord      yScaleFactor,
        WWFixedAsDWord      xScaleFactor);
```

This message notifies the content that the view window's scale factor has changed. The content will set its *VCNI_scaleFactor* field to the passed values.

Source: Unrestricted—typically sent by a GenView to its content object.

Destination: Any VisContent or Process object acting as the content of a GenView.

Parameters:		
	viewWindow	The window handle of the GenView's window.
	yScaleFactor	The new vertical scale factor.
	xScaleFactor	The new horizontal scale factor.

Return: Nothing.

Interception: Any content that is managing large documents will probably need to subclass this message and apply the proper translations for the 32-bit coordinates.

■ MSG_META_CONTENT_VIEW_WIN_OPENED

```
void    MSG_META_CONTENT_VIEW_WIN_OPENED(
        word                viewWidth,
        word                viewHeight,
        WindowHandle        viewWindow);
```

This message notifies the content that the view's window has been created and is being put on the screen. This message will be followed by MSG_META_EXPOSED, so the content should not draw anything here.

Source: Unrestricted—typically sent by a GenView to its content object.

Destination: Any VisContent or Process object acting as the content of a GenView.

Parameters:		
	viewWidth	The new window's initial width.
	viewHeight	The new window's initial height.
	viewWindow	The window handle of the GenView's window.

Return: Nothing.

Interception: A content may wish to subclass this message if it needs to initialize data before the view's window is actually on the screen.

■ MSG_META_CONTENT_VIEW_OPENING

```
void       MSG_META_CONTENT_VIEW_OPENING(
           optr   view);
```

This message notifies the content that the view window is being put on the screen. Although the window will usually be fully realized by the time the content handles this message, the content should not draw anything in this handler. Because the view and content are often in different threads, a context switch could have occurred and the window might not be fully realized. This message will be followed by a MSG_META_EXPOSED indicating that the visible tree can be drawn and that the window is fully opened.

Source: Unrestricted—typically sent by a GenView to its content object.

Destination: Any VisContent or Process object acting as the content of a GenView.

Parameters: *view* The optr of the GenView.

Return: Nothing.

Interception: A content may wish to subclass this message to initialize data before the view window is fully opened.

■ MSG_META_CONTENT_VIEW_SIZE_CHANGED

```
void       MSG_META_CONTENT_VIEW_SIZE_CHANGED(
           word                viewWidth,
           word                viewHeight,
           WindowHandle        viewWindow);
```

This message is sent to the content whenever the view's size changes for any reason. The passed height and width will be stored in the content's *VCNI_viewHeight* and *VCNI_viewWidth* fields.

Source: Unrestricted—typically sent by a GenView to its content object.

Destination: Any VisContent or Process object acting as the content of a GenView.

Parameters:	*viewWidth*	The new width of the view window.
	viewHeight	The new height of the view window.
	viewWindow	The window handle of the GenView's window.

Return: Nothing.

Interception: Any content that is managing large documents may need to subclass this message to apply translations for 32-bit coordinates.

■ MSG_META_CONTENT_VIEW_CLOSING

```
void        MSG_META_CONTENT_VIEW_CLOSING();
```

This message indicates to the content that the view window is being shut down. The content should remove the visible tree from the screen and should prepare itself for the window to be closed.

Source: Unrestricted—typically sent by a GenView to its content object.

Destination: Any VisContent or Process object acting as the content of a GenView.

Interception: A content may subclass this message if it wants to do additional things when the view is taken off the screen.

■ MSG_META_CONTENT_VIEW_WIN_CLOSED

```
void        MSG_META_CONTENT_VIEW_WIN_CLOSED(
            WindowHandle viewWindow);
```

This message indicates that the view's window has been shut down, taken off the screen, and destroyed. The content responds to this message by discarding the window handle stored in its *VCNI_window* field. The content should already have removed itself from the screen when it received an earlier MSG_VIS_CLOSE.

Source: Unrestricted—typically sent by a GenView to its content object.

Destination: Any VisContent or Process object acting as the content of a GenView.

Parameters: *viewWindow* The window handle of the GenView's window.

Return: Nothing.

Interception: A content may subclass this message to clean up after the view window is closed (e.g. if the content cached the view's window handle to a global variable, it will need to zero that handle now).

◆Objects

Generic System Classes

26

The Generic System Objects (GenSystem, GenScreen, and GenField) act as the root objects in the GEOS UI. Every application you create will be in a generic tree headed by these objects; however, you will rarely need to interact with these objects.

The system objects are created by GEOS automatically; you will never need to declare instances of these objects within your code. Whenever an application object is instantiated, the system automatically attaches it to the generic and visible trees headed by these objects.

This chapter is meant to show you how these objects work within the GEOS system, what capabilities they automatically provide for you, and what information they contain that may aid in debugging efforts.

26.1 The System Objects

The GenSystem, GenScreen, and GenField objects work to create a platform for your application to run upon. You will never create these objects, and you might never need to communicate with them. Nevertheless, some applications will want to send messages to these objects. Also, while you are debugging code, you may find it useful to examine the instance data of these objects (though you may never change this data). For these reasons, the objects are fully documented here.

When GEOS starts up, it creates a single GenSystem object. This object is the top-most object of a generic tree which contains all GenApplication objects. It also maintains a separate tree which contains the GenScreen objects. Whenever GEOS is running, there will be exactly one GenSystem object. All generic objects which are displayed on any screen will belong to the generic tree headed by the GenSystem. (See Figure 26-1.)

GEOS creates a GenScreen for every screen being used by the system. Each screen is the head of a Visible tree. (GEOS currently does not support multiple screens on a single system; however, future versions may support multiple screens.)

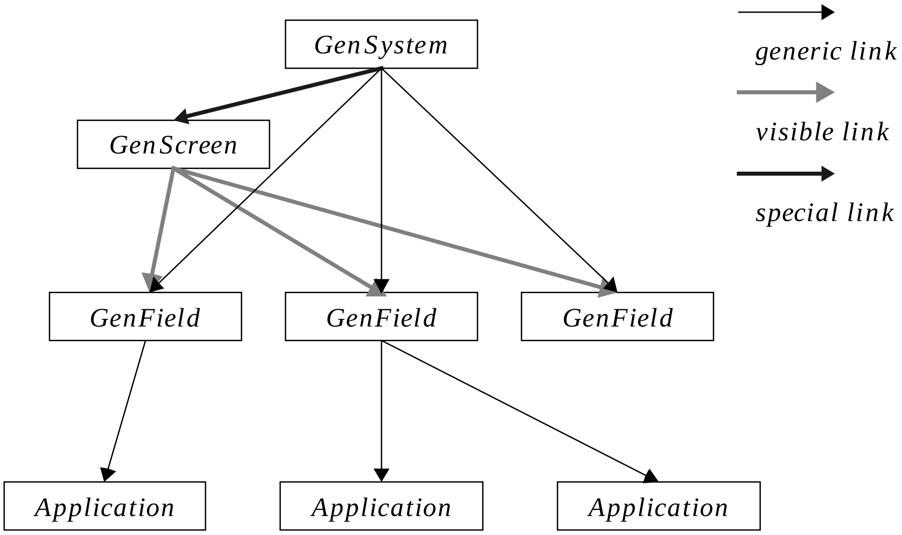


Figure 26-1 *Hierarchy of System Objects*

The GenApplication maintains a set of links to the GenScreen objects; this is distinct from the generic tree headed by the GenSystem. Every GenScreen is the parent of a Visible tree; its children are all GenFields displayed on that screen.

GEOS can provide several different environments, all of which may be running simultaneously on a single machine. For example, Geoworks Ensemble provides “Beginner,” “Intermediate,” and “Advanced” rooms. Each of these environments is represented by a GenField object. Each GenField object is a generic child of the GenSystem. It is also a Visible child of the appropriate GenScreen.

Every Application object is the child of a GenField. When an application is started, its GenApplication is automatically attached to the appropriate GenField, which is generally the GenField which was active when the application was started. An application may transfer itself to a different GenField; this will have the effect of moving the application to a different GEOS environment.

◆Objects

26.2 The GenSystem Object

Whenever GEOS is running, there is a single GenSystem object. This object is the head of the generic tree which contains all generic objects being displayed on any screen. (There may be other generic object trees which are not connected to the GenSystem; however, these objects will have no user interface. For example, every GenDocumentGroup is the head of a separate generic object tree.)

26.2.1 GenSystem Features

The GenSystem object has many responsibilities. Most of these are of no interest to the application. There are a few which applications will want to know about:

- The GenSystem is the head of the generic tree which contains all generic objects which are usable on any screen. This makes it a useful reference point when searching through generic objects during debugging.
- The GenSystem keeps track of the specific UI under which GEOS is running.
- The GenSystem keeps track of the default GenScreen object. Whenever a new GenField is created, it is made a visible child of that GenScreen.
- The GenSystem keeps track of the default GenField object. Whenever an application is started, the application object will be made a generic child of the default GenField.

The GenSystem is also used as a convenient point to alter such system-wide features as the mouse pointer image, system modality, and the layering of windows.

26.2.2 GenSystem Instance Data

GenSystemClass instance data is internal and should never be set or altered by your application. It is provided here for reference, in case you need to examine the data during debugging. However, your code should not examine the data directly; instead, it should use the messages described in section 26.2.3 on page 1457 to examine these fields.

GenSystem itself should also never be subclassed and instances of the class may not be statically defined, nor instantiated by applications or libraries other than the UI library itself.

26.2

Code Display 26-1 GenSystem Instance Data

```
/* Never access or alter these instance data fields.
 * They are for internal use only. */

@instance Handle        GSYI_specificUI;
@instance Handle        GSYI_defaultUI;
@instance optr          GSYI_defaultScreen;
@instance optr          GSYI_defaultField;
@instance @composite    GSYI_screenComp;
```

GSYI_specificUI stores the handle of the specific UI in use by this system object. By default, this specific UI will also be the same specific UI used by all applications underneath this system object.

GSYI_defaultUI stores the handle of the default specific UI to use for the next loaded application. This may be overridden by a GenField (though this is rare). For all intents and purposes, *GSYI_defaultUI* will be the same as *GSYI_specificUI*.

GSYI_defaultScreen stores an optr of a GenScreen. By default, new GenField objects will be made visible children of this GenScreen.

GSYI_defaultField stores the optr of a GenField. By default, new application objects will be made generic children of this GenField. This may be overridden with MSG_GEN_APP_ADD_TO_PARENT, though normal applications will not do this.

GSYI_screenComp stores the optr of the first GenScreen object child. The GenSystem is the head of two object trees. One is a visible tree containing all GenScreen objects; the other is a generic tree containing all application objects (and their generic children). The generic tree is specified by *GI_comp*, as usual.

26.2.3 GenSystem Basics

MSG_GEN_SYSTEM_GET_DEFAULT_SCREEN,
MSG_GEN_SYSTEM_GET_DEFAULT_FIELD,
MSG_GEN_SYSTEM_SET_DEFAULT_FIELD

Some of the **GenSystemClass** instance data can be examined or changed with messages. Applications may find out the current default screen or default field with MSG_GEN_SYSTEM_GET_DEFAULT_SCREEN and MSG_GEN_SYSTEM_GET_DEFAULT_FIELD, respectively. Applications may also set the default field with MSG_GEN_SYSTEM_SET_DEFAULT_FIELD. (Only the system may set the default screen.)

You may retrieve the optr of the current default GenScreen object in use by the system with MSG_GEN_SYSTEM_GET_DEFAULT_SCREEN. You may not change the system's screen object, however.

You may also retrieve the optr of the current default GenField object that GenApplications will be attached to when loaded with MSG_GEN_SYSTEM_GET_DEFAULT_FIELD.

■ MSG_GEN_SYSTEM_GET_DEFAULT_SCREEN

```
optr    MSG_GEN_SYSTEM_GET_DEFAULT_SCREEN();
```

This message returns the optr of the default GenScreen. By default, any GenFields will be made visible children of this GenScreen.

Source: Unrestricted.

Destination: The GenSystem object.

Parameters: None.

Return: The optr of the current default GenScreen object (*GSYI_defaultScreen*).

Interception: Not intercepted.

■ MSG_GEN_SYSTEM_GET_DEFAULT_FIELD

```
optr    MSG_GEN_SYSTEM_GET_DEFAULT_FIELD();
```

This message returns the optr of the default GenField. By default, new applications will be made generic children of this GenField.

Source: Unrestricted.

Destination: The GenSystem object.

Parameters: None.

Return: The optr of the current default GenField object (*GSYI_defaultField*).

Interception: Not intercepted.

■ MSG_GEN_SYSTEM_SET_DEFAULT_FIELD

```
void	MSG_GEN_SYSTEM_SET_DEFAULT_FIELD(
	optr	defaultField);
```

26.2

This message changes the default GenField for the GenSystem. By default, new applications will be made generic children of this GenField.

Source: Unrestricted.

Destination: The GenSystem object.

Parameters: *optr* The optr of the new default GenField object.

Return: Nothing.

Interception: Not intercepted.

26.2.4 Advanced GenSystem Usage

MSG_GEN_SYSTEM_SET_PTR_IMAGE, MSG_GEN_SYSTEM_NOTIFY_SYS_MODAL_WIN_CHANGE, MSG_GEN_SYSTEM_MARK_BUSY, MSG_GEN_SYSTEM_MARK_NOT_BUSY, MSG_GEN_SYSTEM_BRING_GEODE_TO_TOP, MSG_GEN_SYSTEM_LOWER_GEODE_TO_BOTTOM

You may alter the pointer image in use at the system level with MSG_GEN_SYSTEM_SET_PTR_IMAGE. Pass this message the **PointerDef** image to use and the **PointerImageLevel** for the pointer image to represent.

MSG_GEN_SYSTEM_NOTIFY_SYS_MODAL_WIN_CHANGE is sent to the GenSystem whenever it needs to check the status of any system-modal windows. This message is called by the UI whenever it needs to check this information. The message looks for the top window on the screen residing at a window priority of WIN_PRIO_MODAL within a window layer of LAYER_PRIO_MODAL; it then re-directs input to the owning geode's input object. If no system modal window is up, input returns along its normal flow pattern.

◆ Objects

MSG_GEN_SYSTEM_BRING_GEODE_TO_TOP raises a geode's layer to the top, giving the application the focus and target (if it is focusable and/or targetable). Applications will not generally send this message; instead, they will send MSG_GEN_BRING_TO_TOP to the application object, and its handler will send MSG_GEN_SYSTEM_BRING_GEODE_TO_TOP to the GenSystem.

MSG_GEN_SYSTEM_LOWER_GEODE_TO_BOTTOM lowers a geode's layer to the bottom, releases the focus and target from the application (if it had them), and then determines the most suitable geode to grant the focus and target exclusives to. Again, applications will not generally send this message; instead, they will send MSG_GEN_LOWER_TO_BOTTOM to the application object, and its handler will send MSG_GEN_SYSTEM_LOWER_GEODE_TO_BOTTOM to the GenSystem.

■ MSG_GEN_SYSTEM_SET_PTR_IMAGE

```
void    MSG_GEN_SYSTEM_SET_PTR_IMAGE(
        optr                 ptrImage,
        PtrImageLevel        level);
```

This message alters the system-wide pointer image.

Source: Unrestricted.

Destination: The GenSystem object.

Parameters: *optr* A pointer to a **PointerDef** structure.

level The **PtrImageLevel** to use.

Return: Nothing.

Interception: Not intercepted.

■ MSG_GEN_SYSTEM_NOTIFY_SYS_MODAL_WIN_CHANGE

```
void    MSG_GEN_SYSTEM_NOTIFY_SYS_MODAL_WIN_CHANGE();
```

This message is sent to the system object by the UI when it needs to check the status of any system modal windows.

Source: The UI. You should not send this message yourself.

Destination: The GenSystem object.

Interception: Not intercepted.

■ MSG_GEN_SYSTEM_MARK_BUSY

```
void      MSG_GEN_SYSTEM_MARK_BUSY();
```

This message is called by the GenField or GenApplication object while an application is being launched but is not yet on screen. While marked busy, the UI will continue to allow mouse events through. Each message sent to the system object needs a MSG_GEN_SYSTEM_MARK_NOT_BUSY to undo its busy state. Therefore, if multiple MSG_GEN_SYSTEM_MARK_BUSY messages are sent, an equal number of the MSG_GEN_SYSTEM_MARK_NOT_BUSY messages need to be sent to take down the busy cursor.

Source: The UI. You should not send this message yourself.

Destination: The GenSystem object.

Interception: Not intercepted.

■ MSG_GEN_SYSTEM_MARK_NOT_BUSY

```
void      MSG_GEN_SYSTEM_MARK_NOT_BUSY();
```

This message is called by the GenField or GenApplication object when an application no longer needs to mark an application busy that has been brought on-screen.

Source: The UI. You should not send this message yourself.

Destination: The GenSystem object.

Interception: Not intercepted.

■ MSG_GEN_SYSTEM_BRING_GEODE_TO_TOP

```
void      MSG_GEN_SYSTEM_BRING_GEODE_TO_TOP(
          word                geode,
          word                layerID,
          Handle              parentWindow);
```

This message raises a geode's window layer to the top and gives that geode the focus and target (if the geode if focusable/targetable). This message is called from within the UI to implement "autoraise," the automatic raising of a geode and the transferring of the focus and target when clicked upon. This message is also called by the GenApplication's handler for MSG_GEN_BRING_TO_TOP.

Source: Usually the specific UI, in response to an autoraise, or in response to MSG_GEN_BRING_TO_TOP.

Destination: The GenSystem object.

Parameters: *geode* GeodeHandle of the application.

layerID LayerID of window.

parentWindow Handle of the parent window to bring to top.

Return: Nothing.

Interception: Not intercepted.

■ MSG_GEN_SYSTEM_LOWER_GEODE_TO_BOTTOM

```
void    MSG_GEN_SYSTEM_LOWER_GEODE_TO_BOTTOM(
        word                geode,
        word                layerID,
        Handle              parentWindow);
```

This message lowers a geode's window layer to the bottom, releases any focus and target exclusives, and assigns new focus and target exclusives to the most suitable remaining geode. This message is called by the GenApplication's handler for MSG_GEN_LOWER_TO_BOTTOM.

Source: Usually in response to MSG_GEN_LOWER_TO_BOTTOM.

Destination: The GenSystem object.

Parameters: *geode* GeodeHandle of the application.

layerID LayerID of the window.

parentWindow Handle of the parent window to lower to the bottom.

Return: Nothing.

Interception: Generally not intercepted.

26.3 The GenScreen Object

The GenScreen object is an abstract representation of the video screen in use by the system. Its visible bounds are the bounds of the associated screen's video driver.

Currently, only one screen may be in use at a time, so there will be only one GenScreen at any time. (In the future, multiple screens may be supported.)

Every visible component of your system will therefore be associated with a single GenScreen object.

26.3.1 GenScreen Instance Data

GenScreenClass instance data is internal and should never be set or altered by your application. It is provided here for reference, in case you need to examine the data during debugging.

```
@instance Handle  GSCI_videoDriver;
```

GSCI_videoDriver stores the handle of the current video driver in use by the system. This video driver will be used in building the visible tree beneath this object. You should never set this video driver yourself.

26.4 GenField Objects

The GenField object sets the generic field in which all GenApplications attached below it will reside. There may be several GenFields at the same time; each provides a distinct environment for applications. For example, in Geoworks Ensemble, the Beginner, Intermediate, and Advanced Rooms are GenFields.

26.4.1 GenField Features

GenField objects perform many functions. All of these functions are transparent to the application; indeed, most applications can ignore the GenField's existence. Among other things, the GenField does the following:

- It provides a field on which to display windows and other independent UI objects (such as icons, detached menus, etc.).
- It sets the complexity level (novice, advanced, etc.) for all applications under it.
- It manages the Express menu (if one is present).

26.4.2 GenField Instance Data

GenFieldClass instance data is internal and should never be set or altered by your application. It is provided here for reference, in case you need to examine the data during debugging.

Code Display 26-2 GenField Instance Data

```
/* All of these instance fields are internal. They are listed and described here
 * for background information only. */

@instance GenFieldFlags  GFI_flags = 0;

typedef ByteFlags GenFieldFlags;
#define GFF_DETACHING           0x80
#define GFF_LOAD_BITMAP         0x40
#define GFF_RESTORING_APPS      0x20
#define GFF_NEEDS_WORKSPACE_MENU 0x10

@instance optr          GFI_visParent = 0;
@instance byte          GFI_numDetachedApps = 0;
@instance byte          GFI_numRestartedApps = 0;
@instance byte          GFI_numAttachingApps = 0;
@instance ChunkHandle   GFI_apps = 0;
@instance ChunkHandle   GFI_processes = 0;
@instance ChunkHandle   GFI_genApplications = 0;
@instance byte          GFI_numAppsToCheck = 0;
@instance optr          GFI_notificationDestination = 0;
```

GFI_flags stores the **GenFieldFlags** of the GenField object. These flags are for internal bookkeeping purposes.

GFI_visParent stores the optr of the GenScreen object that acts as this GenField's visible parent. This is typically set just before setting the GenField GS_USABLE (although it may be left to the specific UI to fill in.) During MSG_META_DETACH, this field is cleared; it may not be saved to a state file.

GFI_numDetachedApps stores the number of applications that were detached by the system upon MSG_META_DETACH. These applications and their associated state data will be saved to the *GFI_apps* chunk so that they will be brought up in their previous state.

GFI_numRestartedApps stores the number of applications that have been restarted from state files upon MSG_META_ATTACH. These restarted applications should reflect the applications detached during the previous system shutdown.

GFI_numAttachingApps stores the number of applications currently trying to attach to the system. The system keeps track of these applications in case the system must shutdown during an application's attach cycle.

26.4

GFI_apps stores a chunk array of **AppInstanceReference** structures. This data is used on MSG_META_ATTACH to bring up applications to their previous state.

GFI_processes similarly stores a chunk array of processes in progress under this GenField. This chunk array is saved to state so that these processes may continue when GEOS is restarted.

GFI_genApplications stores the number of GenApplications currently launched within the GenField. This data is not saved to state.

GFI_numAppsToCheck stores the number of applications the UI must check with before shutting down.

GFI_notificationDestination stores the object that should receive the Field's notification messages (MSG_META_FIELD_NOTIFY_DETACH, MSG_META_FIELD_NOTIFY_NO_FOCUS, MSG_META_FIELD_NOTIFY_START_LANUCHER_ERROR).

26.4.2.1 GenField Messages

GenFieldClass provides a variety of messages to communicate with the system object and applications. Most of these messages are internal, and those that are not internal are rarely needed. You may in rare cases find it useful to subclass GenField objects, in which case you may need to intercept some of the following messages.

Background Bitmaps

MSG_GEN_FIELD_RESET_BG, MSG_GEN_FIELD_ENABLE_BITMAP

The GenField object may have a bitmap attached to it, to display in the field below the applications.

Field Start-up and Shutdown

MSG_GEN_FIELD_EXIT_TO_DOS, MSG_GEN_FIELD_ABOUT_TO_CLOSE,
MSG_GEN_FIELD_NO_APPS_RESTORED,
MSG_GEN_FIELD_OPEN_WINDOW_LIST,
MSG_GEN_FIELD_GET_TOP_GEN_APPLICATION,
MSG_GEN_FIELD_GET_LAUNCH_MODEL

MSG_GEN_FIELD_EXIT_TO_DOS is sent when the field should exit the system and return to the DOS prompt. This message is sent to the GenField object so that fields may intercept it and perform their own shutdown maintenance. You should only intercept this if you create a custom GenField.

MSG_GEN_FIELD_ABOUT_TO_CLOSE is sent by a GenField environment application (such as Welcome) to tell the field that it is about to be closed. If 'quitOnClose' is set in the field's .INI file, the GenField will attempt to quit all applications running in that GenField. Otherwise, the GenField does nothing; it waits until all open applications are exited, at which point MSG_META_FIELD_NOTIFY_NO_FOCUS will be sent to the GenField.

MSG_GEN_FIELD_NO_APPS_RESTORED is sent to the GenField object when the Field has been restarted to inform it that no applications have been restarted from the state file.

MSG_GEN_FIELD_OPEN_WINDOW_LIST is sent to the GenField to bring up a window list dialog. Certain specific UIs (such as Presentation Manager) support this. Other specific UIs may ignore this message.

MSG_GEN_FIELD_GET_TOP_GEN_APPLICATION returns the top (visual) GenApplication for the GenField sent this message. MSG_GEN_FIELD_GET_LAUNCH_MODEL retrieves the **UILaunchModel** in use for this field.

■ MSG_GEN_FIELD_EXIT_TO_DOS

```
void    MSG_GEN_FIELD_EXIT_TO_DOS();
```

This message requests the GenField to exit the system and return to DOS. The message is sent to the GenField so that custom fields (such as Welcome) can intercept the message and react accordingly by initiating shutdown procedures.

Source: Usually the UI.

Destination: GenField object.

Interception: May be intercepted if you have a custom GenField object that needs to perform certain shutdown procedures before exiting to DOS (such as shutting down applications).

■ MSG_GEN_FIELD_ABOUT_TO_CLOSE

```
void        MSG_GEN_FIELD_ABOUT_TO_CLOSE();
```

This message is sent by an environment application (like Welcome) to inform a GenField that it is about to be closed. The GenField then has the option of quitting any applications or waiting until the applications themselves are closed by the user. If 'quitOnClose' is set the GenField's .INI file category, then the GenField will quit all open applications. Otherwise, it will ignore the request to quit.

Source: Unrestricted, though usually an environment application resident on top of a GenField object.

Destination: The GenField object.

Interception: May be intercepted if you have a custom GenField.

■ MSG_GEN_FIELD_NO_APPS_RESTORED

```
void        MSG_GEN_FIELD_NO_APPS_RESTORED();
```

This message serves as notification that no processes have been restarted from the state file.

Source: The kernel

Destination: GenField object

Interception: May be intercepted to support custom behavior for a custom GenField object.

■ MSG_GEN_FIELD_OPEN_WINDOW_LIST

```
void        MSG_GEN_FIELD_OPEN_WINDOW_LIST();
```

This message may be sent by any object that wishes to bring up a GenField's window list dialog (if available).

Source: Unrestricted.

Destination: GenField object under a specific UI that supports window list dialogs.

Interception: Generally not intercepted.

◆ Objects

■ MSG_GEN_FIELD_GET_TOP_GEN_APPLICATION

```
optr    MSG_GEN_FIELD_GET_TOP_GEN_APPLICATION();
```

This message returns the GenField's top application object.

Source: Unrestricted.

Destination: GenField object.

Return: optr of top visual GenApplication object.

Interception: Generally not intercepted.

■ MSG_GEN_FIELD_GET_LAUNCH_MODEL

```
word    MSG_GEN_FIELD_GET_LAUNCH_MODEL();
```

This message returns the GenField's **UILaunchModel** in use.

Source: Unrestricted.

Destination: GenField object.

Return: **UILaunchModel** in use by the GenField.

Interception: Generally not intercepted.

26.4

Objects

Index

Objects

Objects